

Seguridad del software

PEC 3

Pentesting de software

UOC - MISTIC

Pablo Riutort Grande

20 de diciembre de 2020

Índice

1. Introducción	3
2. Auditoría	4
2.1. <i>Pre-engagement Interactions</i>	4
2.2. <i>Intelligence Gathering</i>	5
2.3. <i>Threat Modeling</i>	6
2.3.1. Identificación de los activos	7
2.3.2. Definición de la arquitectura	7
2.3.3. Descomposición de la aplicación	7
2.3.4. Identificación de las amenazas	8
2.3.5. Mitigación de las amenazas	11
2.4. <i>Exploitation</i>	13
2.4.1. SQL Injection	14
2.5. <i>Reporting</i>	14
A. app.py	16
B. Templates	17
B.1. index.html	17
B.2. news.html	18
B.3. form.html	19

Listings

1. Modificaciones efectuadas en la aplicación para comprobar el tipo	11
2. Modificaciones efectuadas en la aplicación con arquitectura alternativa	12
3. Declaraciones de las entidades User y News	13
4. Modificación de <i>login()</i> para utilizar la entidad User	13
5. app.py	16
6. index.html	17
7. news.html	18
8. form.html	19

Índice de figuras

1. Esquema de la aplicación	3
2. Página principal	3
3. Página detalle de una noticia	4
4. Página de login	4
5. Ejemplo del uso de Maltego sobre la página uoc.edu	6
6. Ejemplo de netcraft sobre la página uoc.edu	6
7. Sistema de login de la aplicación	7
8. Error en el sistema de login	8
9. Árboles de amenazas	10
10. La página carga correctamente con el artículo dado por la URL	12
11. Vista detalle de una noticia.	14
12. SQL injection en parámetro GET. Se obtiene la información de un usuario en vez de un artículo.	14

Índice de cuadros

1. Documentación de amenazas	10
2. Valoración DREAD de SQL injection	11

1. Introducción

Para la realización de este ejercicio haremos el análisis de una supuesta página web de noticias protegida por autenticación por nombre de usuario.

El sistema de login es el componente destinado a la autenticación y, a su vez, actúa como punto de entrada en la autenticación [Fig. 1].

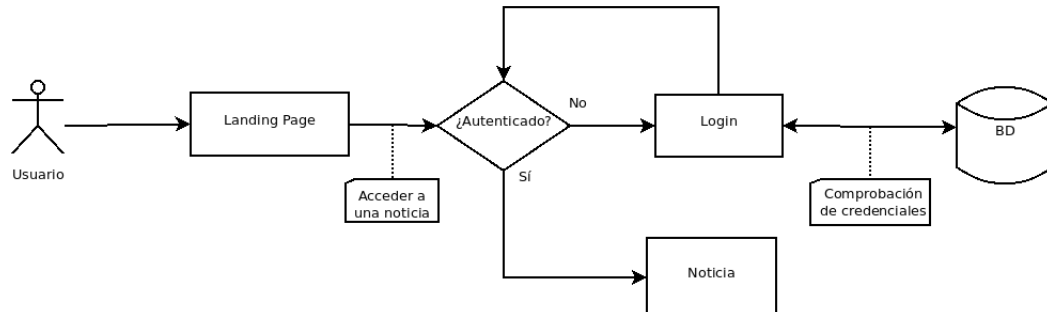


Figura 1: Esquema de la aplicación

El sistema se compone de distintas secciones:

- Página principal: Se muestra un resumen de las noticias que contiene el sistema [Fig. 2].
- Página detalle de una noticia [Fig. 3].
- Página de login [Fig. 4].

Se encuentra adjunto un programa que gestiona el login y la vista de artículos de noticias [Ver A]. Este hace la autenticación por nombre de usuario contra una base de datos y redirecciona a la noticia detalle si existen las credenciales o, en caso contrario, vuelve a la página de login.

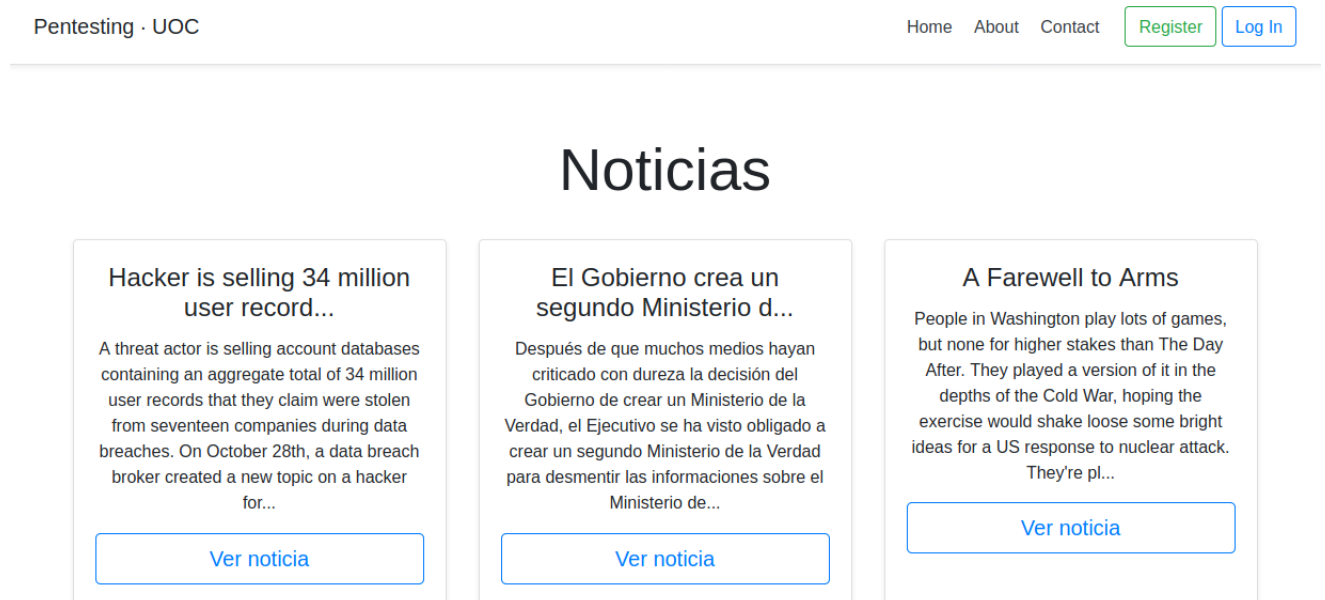


Figura 2: Página principal

A Farewell to Arms

2020-11-22 10:08:14

People in Washington play lots of games, but none for higher stakes than The Day After. They played a version of it in the depths of the Cold War, hoping the exercise would shake loose some bright ideas for a US response to nuclear attack. They're playing it again today, but the scenario has changed - now they're preparing for information war. The game takes 50 people, in five teams of ten. To ensure a fair and fruitful contest, each

Figura 3: Página detalle de una noticia

Log In

Figura 4: Página de login

2. Auditoría

Para la auditoría de este sistema se ha seguido de manera libre el PTES (*Penetration Testing Execution Standard*) [1]. A continuación se describen los distintos puntos de los que se compone el estándar de una manera más detallada.

2.1. *Pre-engagement Interactions*

El primer paso de PTES consiste en establecer las acciones previas al pentesting. Se define el alcance, límites y tiempo y otras restricciones que ayudan a delimitar el ejercicio de pentesting.

En la PTES *guideline* nos ofrecen un cuestionario útil para nuestro escenario de aplicación web [2]:

- **¿Cuántas aplicaciones van a ser tratadas?**

En nuestro caso nos centramos en una única aplicación, la comentada anteriormente en el apartado de introducción [Ver 1].

- **¿Cuántos sistemas de login van a ser tratados?**

Nos centramos en un único sistema de login también comentado en el apartado de introducción [Ver 1].

- **¿Cuántas páginas van a ser tratadas?**

En esta aplicación existen 3 páginas: la de landing, la del detalle de una noticia o artículo y la de login. Sin embargo, para simplificar, solo la de login será tratada en este ejercicio.

- **¿Será el código fuente de la aplicación proporcionado?**

Sí. Queda adjunto en el anexo [A].

- **¿Habrá algún tipo de documentación?**

No.

- **¿Habrá algún tipo de análisis estático?**

Puesto que se proporciona el código fuente, se estudiará.

- **¿El cliente quiere ataque de fuzzing?**

No.

- **¿El cliente quiere tests basados en roles para la aplicación?**

No.

- **¿El cliente quiere escaneado de credenciales sobre aplicaciones web?**

No.

El objetivo de este test es el de penetrar en el sistema sin credenciales de usuario y extraer información privilegiada.

2.2. *Intelligence Gathering*

Este paso consiste en desarrollar una tarea de reconocimiento contra el objetivo para obtener la mayor cantidad de información posible para luego utilizarse durante el análisis de vulnerabilidades y las fases de explotación [3]. Cuanta más información seamos capaces de recolectar durante esta fase más vectores de ataque se podrán identificar.

Existe la gestión de recolección de inteligencia llamada OSINT que consiste en recopilar información disponible públicamente, analizarla y tratarla para convertirla en un recurso. Este proceso nos permite determinar los puntos de entrada en la aplicación. Concretamente, podemos utilizar la guía de OWASP para el *testing information gathering* [4] y software de reconocimiento como Maltego, que consiste en una herramienta OSINT para hacer un análisis gráfico para tareas de investigación [5] [Fig. 5], o Netcraft que también sirve para realizar tareas OSINT [6] [Fig. 6].

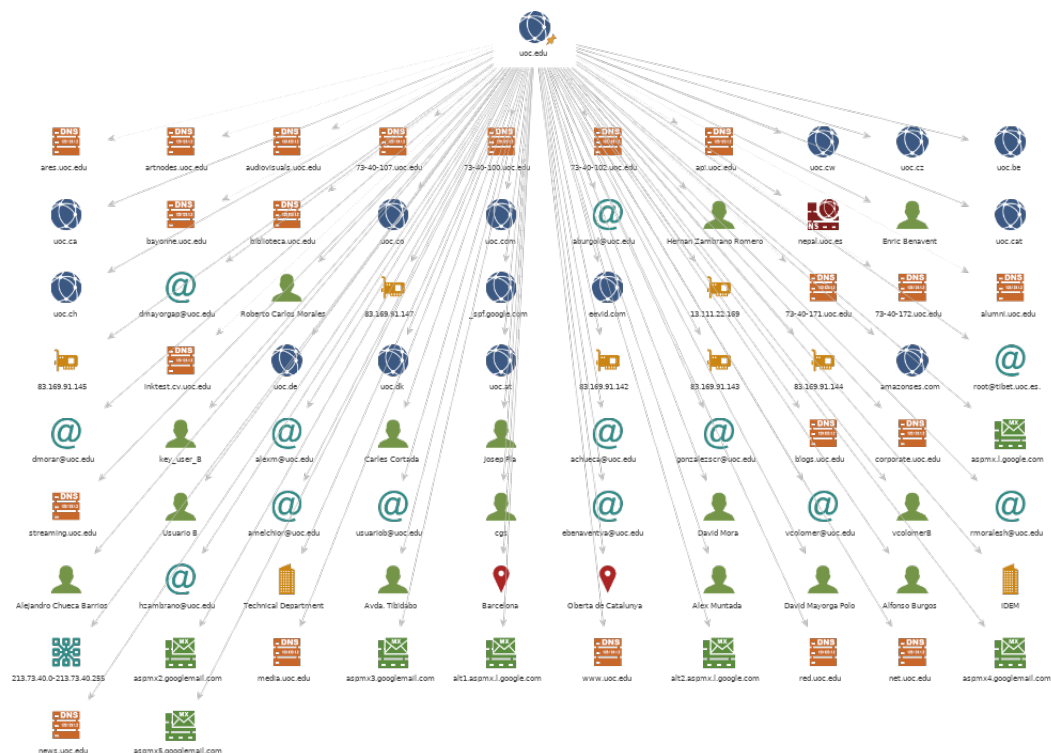



Figura 5: Ejemplo del uso de Maltego sobre la página uoc.edu

Background

Site title	UOC	Date first seen	March 2001
Site rank	85355	Netcraft Risk Rating	Not Present
Description	La UOC es una universidad online que ofrece cursos de grados, posgrados, másters e idiomas.		Primary language
			Spanish

Network

Site	http://www.uoc.edu	Domain	uoc.edu
Netblock Owner	UOC Data Network	Nameserver	tibet.uoc.es
Hosting company	uoc.es	Domain registrar	unknown
Hosting country	 ES	Nameserver organisation	unknown
IPv4 address	213.73.40.242	Organisation	unknown
IPv4 autonomous systems	AS15633	DNS admin	root@tibet.uoc.es
IPv6 address	Not Present	Top Level Domain	Educational entities (.edu)
IPv6 autonomous systems	Not Present	DNS Security Extensions	unknown
Reverse DNS	73-40-242.uoc.es		

IP delegation

IPv4 address (213.73.40.242)			
IP range	Country	Name	Description
0.0.0.0-255.255.255.255	N/A	IANA-BLK	The whole IPv4 address space

Figura 6: Ejemplo de netcraft sobre la página uoc.edu

2.3. Threat Modeling

El modelado de amenazas se requiere para una correcta ejecución del pentest. Se trata de una técnica que identifica las distintas amenazas, ataques o vulnerabilidades y sus contramedidas, de tal forma que se pueda comprender a qué riesgos se ve sometida la aplicación.

Este proceso consiste en evaluar el diseño de la aplicación de manera metódica para encontrar esos fallos de seguridad que puedan ser explotados por el pentester.

El proceso del modelado de amenazas se compone de los siguientes pasos:

2.3.1. Identificación de los activos

Consiste en los activos que se deben proteger. En nuestro caso son tanto las credenciales de usuario como el contenido completo de los artículos de noticia.

2.3.2. Definición de la arquitectura

La arquitectura ha sido brevemente comentada en el apartado de Introducción [Ver 1], sin embargo, se puede concretar en algunos aspectos tecnológicos que puedan ser relevantes para la aplicación.

Esta aplicación ha sido desarrollada con:

- Flask 1.1.2 como framework para gestionar la aplicación [7].
- Flask-MySQL 1.5.1 como conector a la base de datos.
- Docker version 19.03.13, build 4484c46d9d como herramienta de containerización de la infraestructura de base de datos
- MySQL Community Server 8.0.22 como gestor de base de datos [8].
- Bootstrap v4 como framework de templates [9].

2.3.3. Descomposición de la aplicación

Siguiendo el esquema de la aplicación presentado anteriormente en la Introducción [1], podemos descomponer la aplicación en el siguiente diagrama [Fig. 7]:

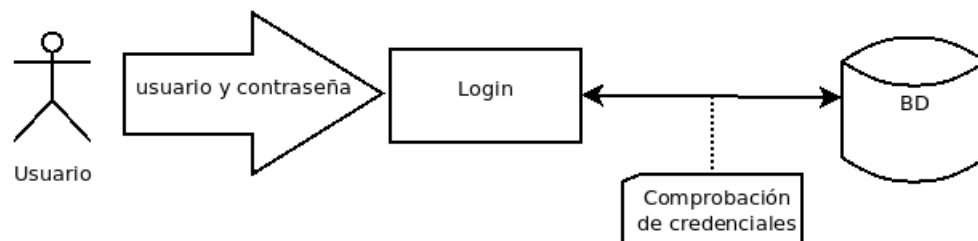


Figura 7: Sistema de login de la aplicación

El flujo de datos es el siguiente:

1. El usuario accede a la página web.
2. El sistema redirige eventualmente a la página de login.
3. El usuario introduce sus credenciales conformado por el par de usuario y contraseña en formulario.
4. El formulario se manda al sistema por POST.
5. El sistema comprueba que esas credenciales se encuentran en base de datos. Construye una query a base de datos comprobando que los credenciales se encuentran en la base de datos y que pertenecen a la misma fila.
6. El sistema devuelve al usuario a la página principal si se ha autenticado correctamente y devuelve un mensaje de error por pantalla [Fig. 8].

Por tanto, el punto de entrada es el formulario de login y este espera el perfil de un usuario autenticado en el sistema.

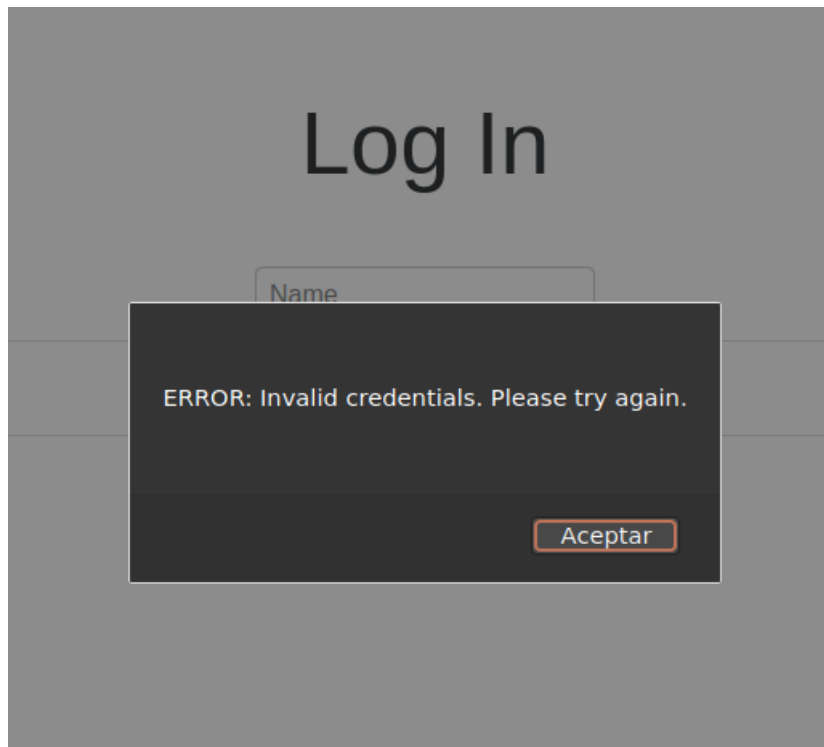


Figura 8: Error en el sistema de login

2.3.4. Identificación de las amenazas

Para nuestro ejercicio y puesto que disponemos del código de la aplicación sabemos de antemano que este sistema es vulnerable a ataques de SQL injection a través del formulario de login.

Ahora se identifican las distintas amenazas que puedan comprometer al sistema. Para ello se lleva a cabo un estudio dividido en 3 partes: Identificación, Documentación y Valoración.

Identificación

Existen varias técnicas de identificación de amenazas, se estudiarán varios enfoques básicos aplicados a nuestro sistema.

■ STRIDE.

STRIDE es un sistema de modelado de amenazas creado por Microsoft. Para aplicarlo, hay que descomponer el sistema en componentes relevantes y analizar su susceptibilidad a amenazas y cómo mitigarlas [10]. La palabra STRIDE es un acrónimo que nos permite clasificar con cada una de sus palabras en distintas categorías:

- *Spoofing*: El atacante puede hacerse pasar por otro usuario. Ejemplo: Un ataque de phishing para que un usuario introduzca los credenciales en un site falso.
- *Tampering*: Modificación de los datos. Ejemplo: Comprometer la integridad de los mensajes
- *Repudiation*: Negación por parte del usuario de haber llevado a cabo una acción en el sistema.
- *Information Disclosure*: Filtrado de información no deseado. Ejemplo: Mensajes descriptados a través de la red.
- *Denial of Service*: El servicio queda inaccesible por parte de usuarios legítimos debido a un atacante. Ejemplo: El sistema es inundado a peticiones hasta que el servidor cae.
- *Elevation of Privileges*: El atacante consigue unos permisos de mayor privilegios a los que posee su usuario. Ejemplo: El atacante cambia su usuario de grupo o rol en el sistema.

Esta técnica nos permite realizar las siguientes preguntas de manera guiada:

¿Puede un usuario no autorizado visualizar datos confidenciales?

Mediante phishing o también con SQL injection, un usuario podría obtener las credenciales de otro usuario y hacerse pasar por este. Las credenciales en esta base de datos se guardan en texto plano y una vez obtenidas se podría visualizar el contenido de la página en su completitud, lo cual es algo que no está permitido.

¿Podría un usuario con privilegios solo de lectura modificar registros en la base de datos?

Sí, se podrían modificar datos mediante SQL injection.

¿Podría un usuario utilizar algún componente para elevar sus privilegios a los de administrador?

Dado que la respuesta a la pregunta anterior ha sido afirmativa, podemos concluir que esta también lo es ya que un usuario mediante un ataque de SQL injection podría modificar sus privilegios a los de administrador.

■ Amenazas clasificadas.

Se clasifican e identifican amenazas constantemente. Existen varios grupos dedicados a la búsqueda e identificación de vulnerabilidades de todo tipo de software que publican rigurosamente sus descubrimientos y hallazgos. Se pueden consultar listados de amenazas comunes y típicos que correspondan a la tecnología y al tipo de aplicación que nos concierne, por ejemplo, el OWASP mantiene un ranking de las top 10 amenazas de las aplicaciones web que podríamos consultar para este caso [12]:

1. **Injection:** Inyecciones SQL, NoSQL, LDAP, etc. ocurren cuando datos no saneados se envían a un intérprete como parte de un comando o una query.
2. **Autenticación defectuosa:** Ocurre cuando la autenticación y la gestión de la sesión están implementados incorrectamente permitiendo a los atacantes obtener passwords o tokens de sesión.
3. **Exposición de datos sensibles:** Muchas aplicaciones web y APIs no protegen correctamente sus datos sensibles. Estos pueden ser sustraídos o modificados por atacantes para realizar fraudes, robo de identidad u otros crímenes.
4. **Entidades XML Externas:** Existen muchos procesadores de XML mal configurados que evalúan referencias externas a otras entidades. Estas entidades se pueden utilizar para hacer escáner de puertos, ejecución remota de código y *DoS attack*.
5. **Control de acceso defectuoso:** Las restricciones de lo que un usuario puede o no puede hacer muchas veces no se aplican correctamente. Los atacantes explotan esta vulnerabilidad para acceder a lugares o funcionalidades restringidas para su tipo de usuario.
6. **Mala configuración de la seguridad:** Es el resultado de que la configuración por defecto del software en cuestión resulta ser insegura. Un ejemplo es el de mensajes de errores con mucha información o cabeceras HTTP mal configuradas.
7. **Cross-Site Scripting (XSS):** Suceden cuando una aplicación muestra datos de origen desconocido en una nueva página sin la validación adecuada permitiendo así al atacante ejecutar scripts en el explorador de la víctima y secuestrar su sesión.
8. **Deserialización insegura:** Provoca ataques de ejecución remota.
9. **Utilización de componentes con vulnerabilidades conocidas:** Muchas librerías y otros componentes tienen los mismos privilegios. Las aplicaciones que utilicen componentes con vulnerabilidades conocidas pueden ser explotadas.
10. **Monitorización y logging insuficientes:** Cuando hay una falta de eso o una integración ineficiente a respuesta de incidentes los atacantes pueden permanecer en el sistema y saltar de este a otros.

De este ranking podemos asegurar que hemos identificado al menos una vulnerabilidad que nos afecta que es la de la inyección SQL.

Árboles de amenazas

Los árboles de amenazas descomponen una amenaza en forma de árbol de manera que las distintas ramificaciones corresponden a formas en las que un atacante puede explotar dicha amenaza. En nuestro caso sabiendo que el punto de entrada es el formulario de login podríamos confeccionar un árbol de esta forma [Fig. 9]:

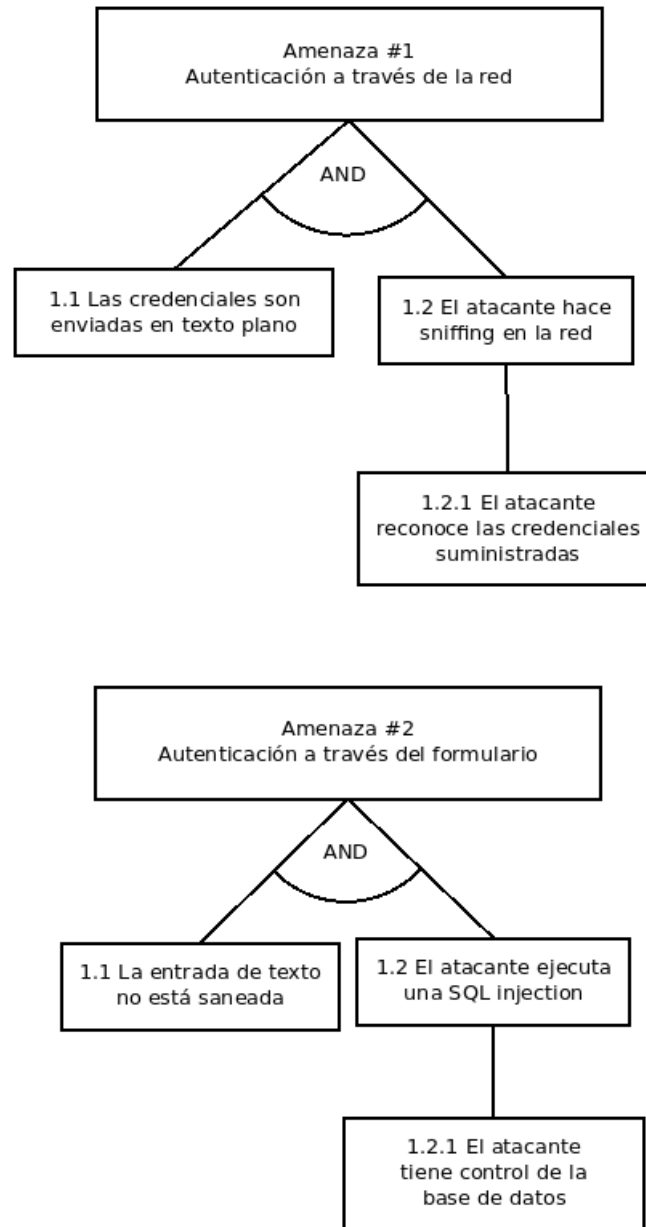


Figura 9: Árboles de amenazas

Documentación

Este paso consiste en documentar las amenazas siguiendo una plantilla similar a [Cuadro 1]:

Descripción de la amenaza	Objetivo de la amenaza	Nivel de riesgo	Técnicas de ataque	Contramedidas
Formulario vulnerable a entrada maliciosa	Leer y modificar BBDD	Muy alto	SQL injection	Saneamiento de la entrada

Cuadro 1: Documentación de amenazas

Valoración Una vez documentadas podemos pasar a valorarlas cada una de ellas para priorizarlas y se hará una clasificación en orden descendiente. La valoración seguirá la siguiente fórmula:

$$Riesgo = Probabilidad \times Daño\ potencial$$

Donde los rangos de Probabilidad y Daño potencial son de 0 a 10 y el rango de Riesgo es de 0 a 100. Supongamos que para nuestra amenaza de SQL injection damos un valor del daño potencial de 10 y una probabilidad de 4, entonces tendremos una valoración del riesgo de $4 \times 10 = 40$.

Alternativamente, podemos evaluar el riesgo utilizando el método DREAD. DREAD vuelve a ser un acrónimo de:

- **Daño potencial:** Magnitud del daño que puede causar la amenaza.
- **Reproductibilidad:** Cómo de fácil es de reproducir.
- **Explotabilidad:** Cómo de fácil es de explotar.
- **Usuarios afectados (*Affected Users*):** Número de usuarios afectados.
- **Detectabilidad:** Cómo de fácil es de encontrar dicha vulnerabilidad.

Cada una de estas palabras corresponde a una categoría cuyos valores pueden ser alto, medio o bajo cuantificados por 3, 2 y 1. El sumatorio de estos valores determina el riesgo total y su clasificación. En el caso de una inyección SQL tendremos algo como [Cuadro 2]:

Amenaza	D	R	E	A	D	Total	Clasificación
SQL injection	3	3	3	3	2	14	Alto

Cuadro 2: Valoración DREAD de SQL injection

2.3.5. Mitigación de las amenazas

Con tal de reducir el impacto de las amenazas identificadas es el momento de llevar a cabo un plan de mitigación de las mismas. En general, existen 4 opciones para mitigarlas:

1. No hacer nada: En general no es una buena solución. Al final el problema tendrá que ser afrontado y solventado.
2. Informar a los usuarios: Se informa a los usuarios del problema que se está tratando y se deja al usuario la decisión de utilizar el programa o no.
3. Eliminar el problema: Quitar la funcionalidad y refactorizarla en próximas versiones.
4. Solucionar el problema: Suele ser la mejor opción y también la más difícil. Consiste en hacer una selección de los recursos necesarios y poner a trabajar a diferentes responsables.

En nuestra aplicación podemos observar que el error se puede mitigar de varias formas, a continuación se pasa a explicar algunas de ellas.

Comprobación del tipo

Una manera de securizar la aplicación podría ser comprobando que, efectivamente, el id del artículo es del tipo correspondiente al que tenemos en base de datos: un entero.

```

1 @app.route("/news")
2 def news():
3     if "user" not in session:
4         return redirect(url_for("login"))
5     article_id = int(request.args.get("id"))
6     article = query_news(article_id)[0]

```

Listing 1: Modificaciones efectuadas en la aplicación para comprobar el tipo

Concretamente se muestra este error:

```

1 ValueError: invalid literal for int() with base 10: '3 union select accountid,email,name
  ,password from users'
2 127.0.0.1 - - [09/Dec/2020 01:34:35] "GET /news?id=3%20union%20select%20accountid,email,
  name,password%20from%20users HTTP/1.1" 500 -

```

Arquitecturas alternativas

Se puede optar por otra arquitectura que mitigue la vulnerabilidad del parámetro GET; por ejemplo, en nuestro caso podríamos optar con obtener el id del artículo como parte de la URL tal que así:

```
1 http://localhost:5000/news/1/
```

Siendo “1” el id del artículo y pasando a formar parte de la URL. Esto podría conseguirse modificando la función que obtiene el id del artículo para que prescindiera del parámetro GET [Ver 2].

```
1 @app.route("/news/<int:article_id>/")
2 def news(article_id):
3     if "user" not in session:
4         return redirect(url_for("login"))
5     article = query_news(article_id)[0]
6     return render_template("news.html", article=article)
```

Listing 2: Modificaciones efectuadas en la aplicación con arquitectura alternativa



Figura 10: La página carga correctamente con el artículo dado por la URL

Uso de ORMs

En los últimos años el uso del *Object Relational Mapping* (ORM) en el desarrollo web se ha popularizado. Esta técnica consiste en utilizar el paradigma de programación orientado a objetos (POO) para crear una clase que represente una entidad en la base de datos. Las aplicaciones orientadas a objetos consiguen la persistencia utilizando sistemas de bases de datos relacionales de tal forma que se relacionan los objetos a tablas [13] y se elimina el uso de queries en raw para acceder a base de datos añadiendo una capa de abstracción.

En este proceso de securización se ha utilizado la librería de SQLAlchemy para relacionar los objetos declarados en la aplicación a tablas de una base de datos.

```
1 from os import getenv
2 from datetime import datetime, timedelta
3 from flask import flash, Flask, redirect, render_template, request, session, url_for
4 from flask_sqlalchemy import SQLAlchemy
5
6
7 app = Flask(__name__)
8 app.secret_key = bytes(getenv("PENTESTING_SECRET"), encoding="utf-8")
9 app.permanent_session_lifetime = timedelta(days=365)
10
11 app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///identidades.sqlite"
12 db = SQLAlchemy(app)
13
14
15 class User(db.Model):
16     account_id = db.Column(db.Integer, primary_key=True)
17     email = db.Column(db.Text, nullable=False)
18     name = db.Column(db.Text, nullable=False)
19     password = db.Column(db.Text, nullable=False)
20
21     def __repr__(self):
22         return f"<User {self.username}>"
23
24
25 class News(db.Model):
26     id = db.Column(db.Integer, primary_key=True)
27     title = db.Column(db.Text, nullable=False)
28     body = db.Column(db.Text, nullable=False)
29     datetime = db.Column(db.DateTime, default=datetime.utcnow)
30
31     def __repr__(self):
32         return f"<Article {self.title}>"
```

Listing 3: Declaraciones de las entidades User y News

Las funciones que anteriormente hacían uso de queries a base de datos ahora pueden servirse de los objetos relacionales para hacer consultas a base de datos o escribir en ella.

```
1 if request.path == "/login" or new_register:
2     user = User.query.filter_by(name=name, password=password)
3     if not user:
```

Listing 4: Modificación de *login()* para utilizar la entidad User

Stored Procedures

Stored Procedure en SQL es un código que se puede guardar en la base de datos para ser reutilizado. En el caso de que tengamos que escribir una query una y otra vez podemos optar por crear un Stored Procedure para llamarlo y que sea este quien ejecute la query [14].

Podemos utilizar esta técnica para la página de noticias cuando selecciona un artículo de la base de datos y securizar así frente a la inyección SQL ya que un Stored Procedure nos permite definir parámetros y así delimitar lo que podemos pasar por id de artículo.

2.4. Exploitation

Una vez hecho el modelado de amenazas y el análisis de las mismas podemos ejecutar la fase de explotación de las mismas. Esta fase está muy fuertemente relacionada con la anterior puesto que cuanto mejor y más exhaustivo sea el análisis más sencilla y directa será esta fase. Esta fase consiste en hacer

una demostración de lo que podría ocurrir si no se mitigan las amenazas encontradas.

2.4.1. SQL Injection

Para ver una noticia se carga por parámetro GET el id de la misma. Este parámetro es recogido por la aplicación y consutrye la query a base de datos. De esta forma se puede seleccionar el artículo pero al mismo tiempo deja a la aplicación vulnerable a un ataque por SQL Injection.

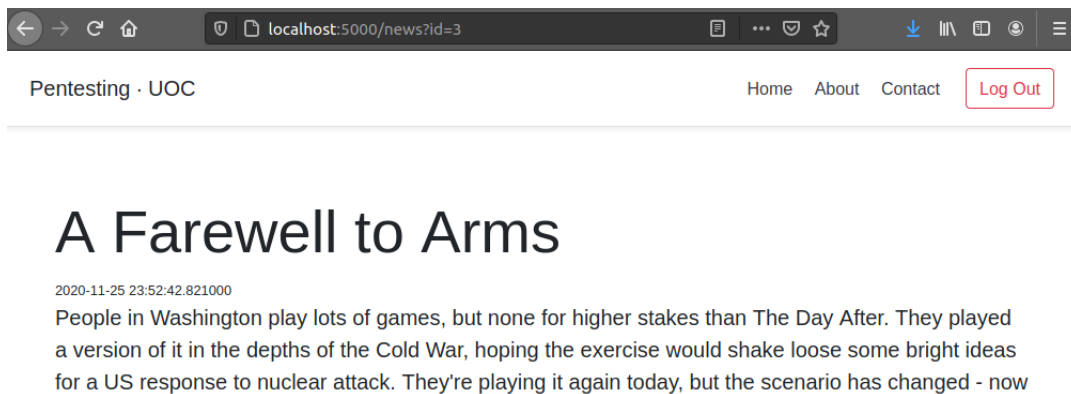


Figura 11: Vista detalle de una noticia.

Un usuario malicioso podría sacar datos de la base de datos construyendo una query aprovechando el parámetro GET:

```
1 http://localhost:5000/news?id=-3%20union%20select%20accountid,email,name,password%20from%20Users
```

Esta dirección será interpretada por la web como la siguiente query a base de datos:

```
1 SELECT * FROM News WHERE Id = -3 UNION SELECT accountid,email,name,password FROM Users;
```

Esta query, por tanto, devolverá información de la tabla de usuarios y la aplicación lo mostrará como si se tratara de una noticia [Fig. 12].



Figura 12: SQL injection en parámetro GET. Se obtiene la información de un usuario en vez de un artículo.

2.5. Reporting

En este paso final se realiza un informe de los hallazgos realizados en el sistema mediante las metodologías empleadas. El informe quedará dividido en 2 partes dirigido a audiencias diferentes: Resumen ejecutivo y el informe técnico.

El informe ejecutivo pretende informar al lector de los objetivos específicos del test de penetración y los resultados hallados a alto nivel. La audiencia de este documento son los que están al cargo de la visión estratégica de los programas de seguridad así como aquellos miembros de la organización que puedan haber sido impactados por las amenazas identificadas. El documento constará de las siguientes partes:

- **Antecedentes:** Se explica el propósito general del test y los detalles de los términos especificados en el primer paso (*Pre-engagement*)
- **Efectividad del test:** Explica cómo han sido capaces los pentesters cumplir los objetivos dentro de los límites establecidos en el primer paso de la auditoría.
- **Perfil de riesgos:** En este apartado se desglosa el perfil de riesgos establecido en el apartado correspondiente de manera general. Se establece una puntuación de riesgo general donde se evalúa al cliente en varios niveles de seguridad dependiendo de la puntuación obtenida.
- **Descubrimientos:** Consiste en una sinopsis de los problemas encontrados en la fase de penetration test en un formato gráfico.
- **Recomendaciones:** Tareas a alto de nivel que son necesarias para mitigar los riesgos identificados y su nivel de esfuerzo para implementarlas.
- **Guía estratégica:** Consiste en un plan de remediación para los riesgos encontrados. Sus niveles de impacto deben ser balanceados contra los objetivos del negocio.

Por otro lado, el informe técnico informa de los aspectos técnicos del test. Describe en detalle el alcance, ataques, impactos y remediación del mismo. Su audiencia, por tanto, será el equipo técnico y consta de las siguientes secciones:

- **Introducción:** Contiene un listado de todos los recursos implicados en el test.
- **Recolección de información:** Es el resultado de la fase de *Intelligence Gathering* dividido en 4 categorías: Inteligencia pasiva, inteligencia activa, inteligencia corporativa e inteligencia del personal.
- **Evaluación de las vulnerabilidades:** Es la clasificación de las vulnerabilidades potenciales que existen en el test. También debe estar presente qué métodos se han utilizado para encontrar dicha vulnerabilidad así como la clasificación de la misma.
- **Confirmación de las vulnerabilidades:** Esta sección contiene el resultado de efectuar la explotación de las vulnerabilidades categorizadas en el apartado anterior. Debe contener, en detalle, los procesos que se han ejecutado para confirmar la vulnerabilidad.
- **Post explotación:** Vincula la capacidad de la explotación con el riesgo real y establece la conexión del impacto real con el cliente al que se le está haciendo el test.
- **Riesgos:** Se trata de dar al cliente la capacidad de monetizar las vulnerabilidades encontradas en el test y sopesar su resolución alineada con los objetivos comerciales. Es una relación transversal de los resultados anteriores con los valores corporativos.
- **Conclusiones**

A. app.py

```
1 from os import getenv
2 from datetime import timedelta
3 from flask import flash, Flask, redirect, render_template, request, session, url_for
4 from flaskext.mysql import MySQL, pymysql
5
6
7 app = Flask(__name__)
8 app.secret_key = bytes(getenv("PENTESTING_SECRET"), encoding="utf-8")
9 app.permanent_session_lifetime = timedelta(days=365)
10
11 app.config['MYSQL_DATABASE_USER'] = 'pentesting'
12 app.config['MYSQL_DATABASE_PASSWORD'] = 'pentesting'
13 app.config['MYSQL_DATABASE_DB'] = 'identidades'
14 database = MySQL(app)
15
16
17 def query(raw_query):
18     try:
19         connection = database.connect()
20         cursor = connection.cursor()
21         cursor.execute(raw_query)
22         return cursor.fetchall()
23     except pymysql.err.Error as error:
24         print(f"[ERROR] {error}")
25     finally:
26         cursor.close()
27         connection.commit()
28         connection.close()
29
30
31 def query_news(article_id=None):
32     raw_query = (
33         f"SELECT * FROM News WHERE Id = {article_id};"
34         if article_id
35         else "SELECT * FROM News;"
36     )
37     return query(raw_query)
38
39
40 @app.route("/")
41 def index():
42     news = query_news()
43     return render_template("index.html", context={"news": news})
44
45
46 @app.route("/news")
47 def news():
48     if "user" not in session:
49         return redirect(url_for("login"))
50     article_id = request.args.get("id")
51     article = query_news(article_id)[0]
52     return render_template("news.html", article=article)
53
54
55 @app.route("/login", methods=["GET", "POST"])
56 def login():
57     if request.method == "POST":
58         new_register = False
59         name = request.form["name"]
60         password = request.form["password"]
61         if request.path == "/login":
62             login_query = f"SELECT * FROM Users WHERE Name='{name}' AND Password='{
password}'"
63             user = query(login_query)
64             if not user:
65                 if new_register:
66                     flash("ERROR: Something went wrong registering user")
67                 else:
68                     flash(f"ERROR: Invalid credentials")
69             else:
70                 session["user"] = True
```



```

71         flash(f"Welcome, {name}")
72         return redirect(url_for("index"))
73     return render_template("form.html")
74
75
76 @app.route("/logout")
77 def logout():
78     session.clear()
79     return redirect(url_for("index"))
80
81
82 if __name__ == "__main__":
83     app.run()

```

Listing 5: Programa conceptual para gestión de login. Vulnerable a heap overflow

B. Templates

B.1. index.html

```

1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
6     <meta name="description" content="Pr ctica pentesting UOC">
7     <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
8     <meta name="generator" content="Jekyll v4.1.1">
9     <title>Pentesting UOC</title>
10
11     <link rel="canonical" href="https://getbootstrap.com/docs/4.5/examples/pricing/">
12
13     <!-- Bootstrap core CSS -->
14     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
15
16     <style>
17       .bd-placeholder-img {
18         font-size: 1.125rem;
19         text-align: middle;
20         -webkit-user-select: none;
21         -moz-user-select: none;
22         -ms-user-select: none;
23         user-select: none;
24       }
25
26       @media (min-width: 768px) {
27         .bd-placeholder-img-lg {
28           font-size: 3.5rem;
29         }
30       }
31     </style>
32   </head>
33   <body>
34     <div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-white border-bottom shadow-sm">
35       <h5 class="my-0 mr-md-auto font-weight-normal">Pentesting UOC</h5>
36       <nav class="my-2 my-md-0 mr-md-3">
37         <a class="p-2 text-dark" href="/">Home</a>
38         <a class="p-2 text-dark" href="#">About</a>
39         <a class="p-2 text-dark" href="mailto:priutortg@uoc.edu">Contact</a>
40       </nav>
41       <div>
42         {% if session['user'] %}
43         <a class="btn btn-outline-danger" href="/logout">Log Out</a>
44         {% else %}
45         <a class="btn btn-outline-success" href="/register">Register</a>
46         <a class="btn btn-outline-primary" href="/login">Log In</a>
47         {% endif %}
48       </div>

```

```

49 </div>
50
51 <div class="pricing-header px-3 py-3 pt-md-5 pb-md-4 mx-auto text-center">
52   <h1 class="display-4">Noticias</h1>
53 </div>
54
55 <div class="container">
56   <div class="card-deck mb-3 text-center">
57     {% for article in context['news'] %}
58     <div class="card mb-4 shadow-sm">
59       <div class="card-body">
60         <h4 class="card-title pricing-card-title">{{ article[1][:40] + '...' if
61 article[1]|length > 40 else article[1] }}</h4>
62         <p>{{ article[2][:250] }}...</p>
63         <a href="/news?id={{ article[0] }}" type="button" class="btn btn-lg btn-
64 block btn-outline-primary">Ver noticia</a>
65       </div>
66     </div>
67     {% endfor %}
68   </div>
69 </body>
</html>

```

Listing 6: Template para la vista principal

B.2. news.html

```

1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
6     <meta name="description" content="">
7     <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
8     <meta name="generator" content="Jekyll v4.1.1">
9     <title>Pentesting    UOC</title>
10
11     <link rel="canonical" href="https://getbootstrap.com/docs/4.5/examples/pricing/">
12
13     <!-- Bootstrap core CSS -->
14     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
15
16     <style>
17       .bd-placeholder-img {
18         font-size: 1.125rem;
19         text-align: middle;
20         -webkit-user-select: none;
21         -moz-user-select: none;
22         -ms-user-select: none;
23         user-select: none;
24       }
25
26       @media (min-width: 768px) {
27         .bd-placeholder-img-lg {
28           font-size: 3.5rem;
29         }
30       }
31     </style>
32   </head>
33   <body>
34     <div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-white border-bottom shadow-sm">
35       <h5 class="my-0 mr-md-auto font-weight-normal">Pentesting    UOC</h5>
36       <nav class="my-2 my-md-0 mr-md-3">
37         <a class="p-2 text-dark" href="/">Home</a>
38         <a class="p-2 text-dark" href="#">About</a>
39         <a class="p-2 text-dark" href="mailto:priortortg@uoc.edu">Contact</a>
40       </nav>
41     </div>

```

```

42     {% if session['user'] %}
43     <a class="btn btn-outline-danger" href="/logout">Log Out</a>
44     {% else %}
45     <a class="btn btn-outline-success" href="/register">Register</a>
46     <a class="btn btn-outline-primary" href="/login">Log In</a>
47     {% endif %}
48 </div>
49 </div>
50
51 <div class="pricing-header px-5 py-3 pt-md-5 pb-md-4 mx-auto">
52   <h3 class="display-4">{{ article[1] }}</h3>
53   <small>{{ article[3] }}</small>
54   <p class="lead">{{ article[2] }}</p>
55 </div>
56
57 </body>
58 </html>

```

Listing 7: Template para la vista de noticias

B.3. form.html

```

1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
6     <meta name="description" content="Pr ctica pentesting UOC">
7     <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
8     <meta name="generator" content="Jekyll v4.1.1">
9     <title>Pentesting    UOC</title>
10
11     <link rel="canonical" href="https://getbootstrap.com/docs/4.5/examples/pricing/">
12
13     <!-- Bootstrap core CSS -->
14     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
15
16     <style>
17       .bd-placeholder-img {
18         font-size: 1.125rem;
19         text-align: middle;
20         -webkit-user-select: none;
21         -moz-user-select: none;
22         -ms-user-select: none;
23         user-select: none;
24       }
25
26       @media (min-width: 768px) {
27         .bd-placeholder-img-lg {
28           font-size: 3.5rem;
29         }
30       }
31     </style>
32   </head>
33   <body>
34     <div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-white border-bottom shadow-sm">
35       <h5 class="my-0 mr-md-auto font-weight-normal">Pentesting    UOC</h5>
36       <nav class="my-2 my-md-0 mr-md-3">
37         <a class="p-2 text-dark" href="/">Home</a>
38         <a class="p-2 text-dark" href="#">About</a>
39         <a class="p-2 text-dark" href="mailto:priortortg@uoc.edu">Contact</a>
40       </nav>
41       <div>
42         <a class="btn btn-outline-success" href="/register">Register</a>
43         <a class="btn btn-outline-primary" href="/login">Log In</a>
44       </div>
45     </div>
46
47     <div class="pricing-header px-3 py-3 pt-md-5 pb-md-4 mx-auto text-center">

```

```

48     {% if request.path == "/register" %}
49     <h1 class="display-4">Registrar usuario</h1>
50     {% else %}
51     <h1 class="display-4">Log In</h1>
52     {% endif %}
53 </div>
54
55 <div class="container">
56     {% if request.path == "/register" %}
57     <form action="/register" method="POST">
58     {% else %}
59     <form action="/login" method="POST">
60     {% endif %}
61         <div class="text-center">
62             <ul class="list-group list-group-flush">
63                 {% if request.path == "/register" %}
64                 <li class="list-group-item">
65                     <input name="email" placeholder="Email" type="email" required>
66                 </li>
67                 {% endif %}
68                 <li class="list-group-item">
69                     <input name="name" placeholder="Name" required>
70                 </li>
71                 <li class="list-group-item">
72                     <input name="password" placeholder="Password" type="password" required>
73                 </li>
74                 <li class="list-group-item">
75                     <button class="btn btn-success" type="submit">Submit</button>
76                 </li>
77             </ul>
78         </div>
79     </form>
80 </div>
81
82 {% with messages = get_flashed_messages() %}
83 {% if messages %}
84 <script>
85     alert("{% messages[0] %}");
86 </script>
87 {% endif %}
88 {% endwith %}
89
90 </body>
91 </html>

```

Listing 8: Template para gestin de login

Referencias

- [1] **High Level Organization of the Standard**
PTES Technical Guideline [Consultado el 1 de diciembre de 2020]
http://www.pentest-standard.org/index.php/Main_Page
- [2] **Pre-engagement**
PTES Technical Guideline [Consultado el 1 de diciembre de 2020]
<http://www.pentest-standard.org/index.php/Pre-engagement>
- [3] **Intelligence Gathering**
PTES Technical Guideline [Consultado el 3 de diciembre de 2020]
http://www.pentest-standard.org/index.php/Intelligence_Gathering#General
- [4] **Testing Information Gathering**
OWASP [Consultado el 4 de diciembre de 2020]
https://wiki.owasp.org/index.php/Testing_Information_Gathering
- [5] **Maltego** [Consultado el 7 de diciembre de 2020]
<https://www.maltego.com/>
- [6] **SpiderFoot HX**
OSINT for Professionals [Consultado el 7 de diciembre de 2020]
<https://www.spiderfoot.net/hx/>
- [7] **Flask.** [Consultado el 7 de diciembre de 2020]
<https://flask.palletsprojects.com/en/1.1.x/>
- [8] **MySQL is a widely used, open-source relational database management system (RDBMS).**
Docker Official Images. [Consultado el 7 de diciembre de 2020]
https://hub.docker.com/_/mysql
- [9] **Bootstrap.** [Consultado el 7 de diciembre de 2020]
<https://getbootstrap.com/>
- [10] **Uncover Security Design Flaws Using The STRIDE Approach**
MSDN Magazine [Consultado el 10 de diciembre de 2020]
<https://web.archive.org/web/20070303103639/http://msdn.microsoft.com/msdnmag/issues/06/11/ThreatModeling/default.aspx>
- [11] **STRIDE Threat Model**
Kenneth Peeples - 2 de diciembre de 2015 [Consultado el 10 de diciembre de 2020]
<https://dzone.com/articles/stride-threat-model>
- [12] **Top 10 Web Application Security Risks**
OWASP [Consultado el 11 de diciembre de 2020]
<https://owasp.org/www-project-top-ten/>
- [13] **Object-Relational Mapping Revisited - A Quantitative Study on the Impact of Database Technology on O/R Mapping Strategies**
Semantic Scholar. [Consulta: 11 de diciembre de 2020]
<https://www.semanticscholar.org/paper/Object-Relational-Mapping-Revisited-A-Quantitative-Lorenz-708ac5e798b7e45b949d42e2f872549a3612e1e2>
- [14] **"SQL Stored Procedures for SQL Server",**
w3schools.com. [Consulta: 28 de noviembre de 2020]
https://www.w3schools.com/sql/sql_stored_procedures.asp