Ejercicio 1

Vulnerabilidades:

1. Introducir credenciales en el código fuente: introducir credenciales o información de autenticación en el código fuente de una aplicación, la cual va a ser usada por muchas personas diferentes, puede propiciar la lectura de dichas credenciales por alguna de estas personas y causar el acceso a los datos confidenciales de la aplicación/empresa. Esto puede llevar a la difusión de secretos empresariales, causando pérdidas financieras y una pérdida de seguridad en dicha app/empresa.
2. Guardar contraseñas en ficheros de texto o usando un algoritmo de fast hashing: guardar contraseñas de esta manera supone un riesgo de seguridad para las aplicaciones de software. Un atacante que consiga acceso a dichas contraseñas, las puede reusar sin necesidad de más ataques o sin requerir mucho más esfuerzo. Estas contraseñas se pueden usar para acceder a las cuentas de usuario, lo que permitiría realizar varias actividades maliciosas.
3. Basar las autorizaciones en buenas decisiones: la autorización del acceso de un usuario a ciertos recursos de una aplicación se debe basar en buenas y robustas decisiones. Es decir, la autorización del acceso se debe permitir si se cumplen ciertos criterios (ej: el usuario tiene el rol y los privilegios necesarios, el usuario ha sido autenticado correctamente, etc). La no verificación del acceso de los usuarios puede comportar importantes riesgos de seguridad.

Bugs:

1. Funciones recursivas infinitas: las funciones/métodos recursivos siempre deben llevar a un caso base que permita parar la recursividad. Una recursividad infinita "romperá" la aplicación.
2. Bucles infinitos: un bucle infinito nunca terminará mientras el programa está funcionando. Por tanto, habrá que "matar" el programa para salir del bucle y no permitirá ejecutar el resto de funciones del programa. Todos los bucles deben incluir una manera de alterar la condición para poder terminar el bucle cuando deje de cumplirse la condición.
3. Acceder a un elemento de un array que causa una ArrayIndexOutOfBoundsException: estos errores ocurren cuando tratamos de acceder a un elemento del array que no existe. Hay que asegurarse, al trabajar con arrays, que manipulamos elementos que estén dentro de los límites del array. Estos errores de excepción pueden provocar comportamientos inesperados o "romper" el programa.

Security hotspot:

1. Usar directorios con permisos "write" para todos. Una aplicación que manipula archivos contenidos en este tipo de directorios se expone a que un usuario malicioso lance un ataque que resulte en el acceso a los archivos, pudiéndolos modificar, corromper o borrar.
2. Incluir secretos en el código fuente: se pueden extraer muy fácilmente cadenas de texto del código fuente de una aplicación, lo que podría llevar a la obtención de dichos secretos. Sobre todo si las aplicaciones son open-source o se distribuyen a muchos usuarios.
3. Usar protocolos como ftp, telnet o http: estos no encriptan los datos transportades, ni tienen capacidad para establecer una conexión autenticada. Un atacante podría leer, modificar o corromper los datos transportados. Estos protocolos exponen a las aplicaciones a muchos riesgos.

Code smell:

1. Un método devuelve un valor fijo: si un método devuelve siempre el mismo valor, es un signo de mal diseño. El comando return del método debe devolver diferentes valor según los parámetros que pasemos al método.
2. Usar “replaceAll” en lugar de “replace”: “replaceAll” consume más recursos que “replace”, por lo que se debe evitar usar “replaceAll” por defecto si no está justificado.
3. La cláusula “default” del switch no aparece al final: por motivos de facilitar la lectura, se recomienda poner la cláusula “default” del switch al final de la estructura de control, para poder identificarlas rápidamente.

Ejercicio 2

```
1  package foo;
2
3  public class Flow {
4
5      private Flow() {
6      }
7
8      public static void main(String[] args) {
9          Object o = getObject();
10         Object o2 = o;
11         System.out.println(o2.toString());
12     }
13
14     private static Object getObject() {
15         return null;
16     }
17
18 }
19
```

SonarLint

Las recomendaciones de SonarLint aparecen en la captura de pantalla.

- Mover el archivo a un paquete con nombre: el archivo ha sido creado en el paquete por defecto. Habría que crear un paquete llamado “foo” y mover el archivo a este paquete.
- Eliminar la clase vacía, escribir su código o convertirla en una interficie.

Aunque no aparezca en las recomendaciones de SonarLint, según la información de su web, el método getObject() sería un “bad smell”, ya que siempre devuelve lo mismo, null. Además, el constructor Flow() debería ser público y tener contenido.