Fortify Audit Workbench

Developer Workbook

Instagram_v1

Table of Contents

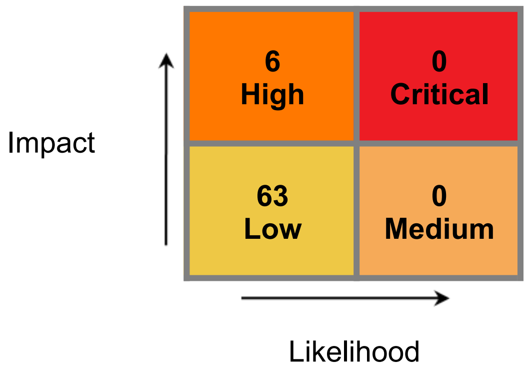
Executive Summary

This workbook is intended to provide all necessary details and information for a developer to understand and remediate the different issues discovered during the Instagram_v1 project audit. The information contained in this workbook is targeted at project managers and developers.

This section provides an overview of the issues uncovered during analysis.

Project Name:	Instagram_v1
Project Version:	
SCA:	Results Present
WebInspect:	Results Not Present
WebInspect Agent:	Results Not Present
Other:	Results Not Present

Issues by Priority



Top Ten Critical Categories

This project does not contain any critical issues

Project Description

This section provides an overview of the Fortify scan engines used for this project, as well as the project meta-information.

Issue Breakdown by Fortify Categories

The following table depicts a summary of all issues grouped vertically by Fortify Category. For each category, the total number of issues is shown by Fortify Priority Order, including information about the number of audited issues.

Category	Fortify Priority (audited/total)				Total Issues
	Critical	High	Medium	Low	
Code Correctness: Class Does Not Implement equals	0	0	0	0 / 1	0 / 1
Command Injection	0	0 / 1	0	0	0 / 1
Path Manipulation	0	0 / 1	0	0	0 / 1
Poor Logging Practice: Use of a System Output Stream	0	0	0	0 / 58	0 / 58
Poor Style: Identifier Contains Dollar Symbol ($)	0	0	0	0 / 1	0 / 1
Poor Style: Value Never Read	0	0	0	0 / 1	0 / 1
Portability Flaw: File Separator	0	0 / 1	0	0	0 / 1
Portability Flaw: Locale Dependent Comparison	0	0 / 3	0	0	0 / 3
System Information Leak: Internal	0	0	0	0 / 2	0 / 2

Results Outline

Code Correctness: Class Does Not Implement equals (1 issue)

Abstract

The `equals()` method is called on an object that does not implement `equals()`.

Explanation

When comparing objects, developers usually want to compare properties of objects. However, calling `equals()` on a class (or any super class/interface) that does not explicitly implement `equals()` results in a call to the `equals()` method inherited from `java.lang.Object`. Instead of comparing object member fields or other properties, `Object.equals()` compares two object instances to see if they are the same. Although there are legitimate uses of `Object.equals()`, it is often an indication of buggy code. **Example 1:**

```
public class AccountGroup
{
    private int gid;

    public int getGid()
    {
        return gid;
    }

    public void setGid(int newGid)
    {
        gid = newGid;
    }
}
...
public class CompareGroup
{
    public boolean compareGroups(AccountGroup group1, AccountGroup group2)
    {
        return group1.equals(group2);   //equals() is not implemented in
AccountGroup
    }
}
```

Recommendation

Verify that the use of `Object.equals()` is really the method you intend to call. If not, implement an `equals()` method or use a different method for comparing objects. **Example 2:** The following code adds an `equals()` method to the example from the Explanation section.

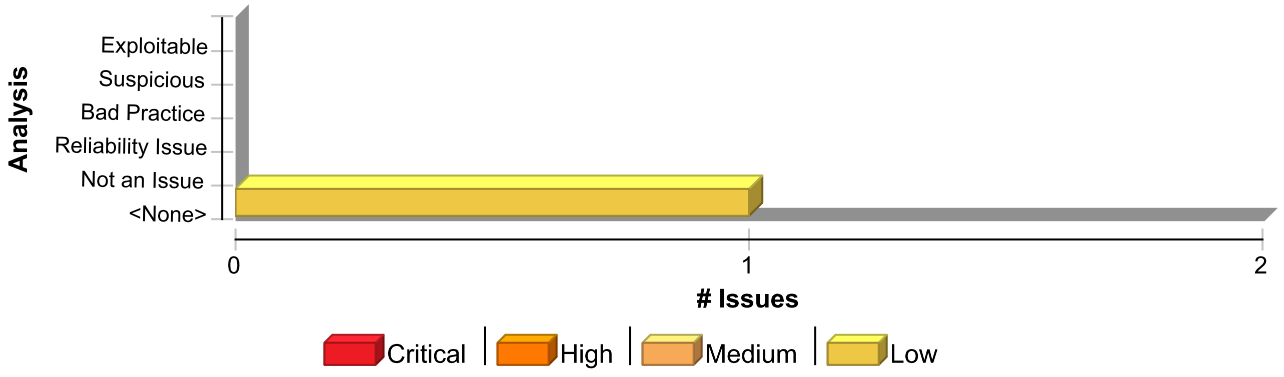
```
public class AccountGroup
{
    private int gid;

    public int getGid()
    {
        return gid;
    }

    public void setGid(int newGid)
    {
        gid = newGid;
    }

    public boolean equals(Object o)
    {
        if (!(o instanceof AccountGroup))
            return false;
        AccountGroup other = (AccountGroup) o;
        return (gid == other.getGid());
    }
}
...
public class CompareGroup
{
    public static boolean compareGroups(AccountGroup group1, AccountGroup
group2)
    {
        return group1.equals(group2);
    }
}
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Class Does Not Implement equals	1	0	0	1
Total	**1**	**0**	**0**	**1**

Code Correctness: Class Does Not Implement equals **Low**

Package: src

src/Perfil.java, line 90 (Code Correctness: Class Does Not Implement equals) **Low**

Issue Details

Kingdom: API Abuse
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: equals

Code Correctness: Class Does Not Implement equals — Low

Package: src

src/Perfil.java, line 90 (Code Correctness: Class Does Not Implement equals) — Low

Enclosing Method: agregarSeguidor()
File: src/Perfil.java:90
Taint Flags:

```
*/
public void agregarSeguidor(Usuario seguidor) {
for (Usuario seguidores : seguidores) {
if (seguidores.equals(seguidor)) {
System.out.println("Ya sigues a " + seguidor.getNombre());
return;
}
```

Command Injection (1 issue)

Abstract

Executing commands from an untrusted source or in an untrusted environment can cause an application to execute malicious commands on behalf of an attacker.

Explanation

Command injection vulnerabilities take two forms: - An attacker can change the command that the program executes: the attacker explicitly controls what the command is. - An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means. In this case, we are primarily concerned with the first scenario, the possibility that an attacker may be able to control the command that is executed. Command injection vulnerabilities of this type occur when: 1. Data enters the application from an untrusted source. 2. The data is used as or as part of a string representing a command that is executed by the application. 3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have. **Example 1:** The following code from a system utility uses the system property `APPHOME` to determine the directory in which it is installed and then executes an initialization script based on a relative path from the specified directory.

```
    ...
    String home = System.getProperty("APPHOME");
    String cmd = home + INITCMD;
    java.lang.Runtime.getRuntime().exec(cmd);
    ...
```

The code in `Example 1` allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property `APPHOME` to point to a different path containing a malicious version of `INITCMD`. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property `APPHOME`, then they can fool the application into running malicious code and take control of the system. **Example 2:** The following code is from an administrative web application designed to allow users to kick off a backup of an Oracle database using a batch-file wrapper around the `rman` utility and then run a `cleanup.bat` script to delete some temporary files. The script `rmanDB.bat` accepts a single command line parameter, which specifies the type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

```
...
String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K
\"c:\\util\\rmanDB.bat "+btype+"&&c:\\util\\cleanup.bat\"")
System.Runtime.getRuntime().exec(cmd);
...
```

The problem here is that the program does not do any validation on the `backuptype` parameter read from the user. Typically the `Runtime.exec()` function will not execute multiple commands, but in this case the program first runs the `cmd.exe` shell in order to run multiple commands with a single call to `Runtime.exec()`. After the shell is invoked, it will allow for the execution of multiple commands separated by two ampersands. If an attacker passes a string of the form `"&& del c:\\dbms\\*.*"`, then the application will execute this command along with the others specified by the program. Because of the nature of the application, it runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well. **Example 3:** The following code is from a web application that provides an interface through which users can update their password on the system. Part of the process for updating passwords in certain network environments is to run a `make` command in the `/var/yp` directory.

```
...
System.Runtime.getRuntime().exec("make");
...
```

The problem here is that the program does not specify an absolute path for make and fails to clean its environment prior to executing the call to `Runtime.exec()`. If an attacker can modify the `$PATH` variable to point to a malicious binary called `make` and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's `make` will now be run with these privileges, possibly giving the attacker complete control of the system. Some think that in the mobile world, classic vulnerabilities, such as command injection, do not make sense -- why would a user attack him or herself? However, keep in mind that the essence of mobile platforms is applications that are downloaded from various sources and run alongside each other on the same device. The likelihood of running a piece of malware next to a banking application is high, which necessitates expanding the attack surface of mobile applications to include inter-process communication. **Example 4:** The following code reads commands to be executed from an Android intent.

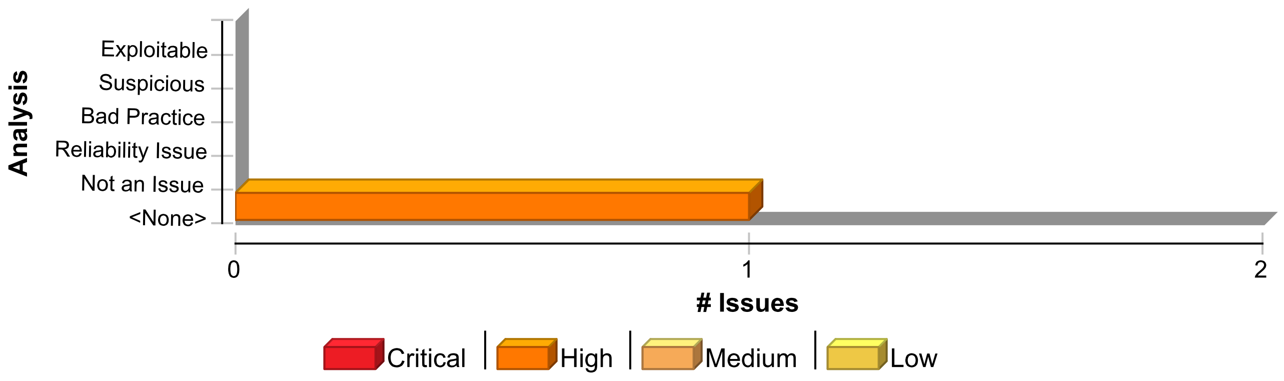
```
...
        String[] cmds = this.getIntent().getStringArrayExtra("commands");
        Process p = Runtime.getRuntime().exec("su");
        DataOutputStream os = new DataOutputStream(p.getOutputStream());
        for (String cmd : cmds) {
                os.writeBytes(cmd+"\n");
        }
        os.writeBytes("exit\n");
        os.flush();
...
```

On a rooted device, a malicious application can force a victim application to execute arbitrary commands with super user privileges.

Recommendation

Do not allow users to have direct control over the commands executed by the program. In cases where user input must affect the command to be run, use the input only to make a selection from a predetermined set of safe commands. If the input appears to be malicious, the value passed to the command execution function should either default to some safe selection from this set or the program should decline to execute any command at all. In cases where user input must be used as an argument to a command executed by the program, this approach often becomes impractical because the set of legitimate argument values is too large or too hard to keep track of. Developers often fall back on blacklisting in these situations. Blacklisting selectively rejects or escapes potentially dangerous characters before using the input. Any list of unsafe characters is likely to be incomplete and will be heavily dependent on the system where the commands are executed. A better approach is to create a whitelist of characters that are allowed to appear in the input and accept input composed exclusively of characters in the approved set. An attacker may indirectly control commands executed by a program by modifying the environment in which they are executed. The environment should not be trusted and precautions should be taken to prevent an attacker from using some manipulation of the environment to perform an attack. Whenever possible, commands should be controlled by the application and executed using an absolute path. In cases where the path is not known at compile time, such as for cross-platform applications, an absolute path should be constructed from trusted values during execution. Command values and paths read from configuration files or the environment should be sanity-checked against a set of invariants that define valid values. Other checks can sometimes be performed to detect if these sources may have been tampered with. For example, if a configuration file is world-writable, the program might refuse to run. In cases where information about the binary to be executed is known in advance, the program may perform checks to verify the identity of the binary. If a binary should always be owned by a particular user or have a particular set of access permissions assigned to it, these properties can be verified programmatically before the binary is executed. Although it may be impossible to completely protect a program from an imaginative attacker bent on controlling the commands the program executes, be sure to apply the principle of least privilege wherever the program executes an external command: do not hold privileges that are not essential to the execution of the command.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Command Injection	1	0	0	1
Total	**1**	**0**	**0**	**1**

Command Injection — High

Package: src

src/Perfil.java, line 48 (Command Injection) — High

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read java.lang.System.in
From: Main.main
File: src/Main.java:36

```
System.out.println(YELLOW + "###### ## ## ###### ## ## ## ########## ## ## ## ## ## ## " + RESET);
System.out.println("-----------------------------------------------------------
System.out.println("-----------------------------------------------------------
Scanner scanner = new Scanner(System.in);
boolean salir = false;
while (!salir) {
System.out.println(PURPLE + "----- MEN ⱡ PRINCIPAL -----" + RESET);
```

Sink Details

Sink: java.awt.Desktop.open()
Enclosing Method: mostrarContenidoPublicaciones()
File: src/Perfil.java:48
Taint Flags: STDIN, TAINTED_PATH

```
String rutaFoto = publicacion.getRutaFoto();
File foto = new File(rutaFoto);
```

Command Injection	High
Package: src	
src/Perfil.java, line 48 (Command Injection)	**High**

```
System.out.println(publicacion.toString());
Desktop.getDesktop().open(foto);
i++;
}
}
```

Path Manipulation (1 issue)

Abstract

Allowing user input to control paths used in file system operations could enable an attacker to access or modify otherwise protected system resources.

Explanation

Path manipulation errors occur when the following two conditions are met: 1. An attacker is able to specify a path used in an operation on the file system. 2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker. **Example 1:** The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as `"../../tomcat/conf/server.xml"`, which causes the application to delete one of its own configuration files.

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

Example 2: The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with adequate privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension `.txt`.

```
fis = new FileInputStream(cfg.getProperty("sub")+".txt");
amt = fis.read(arr);
out.println(arr);
```

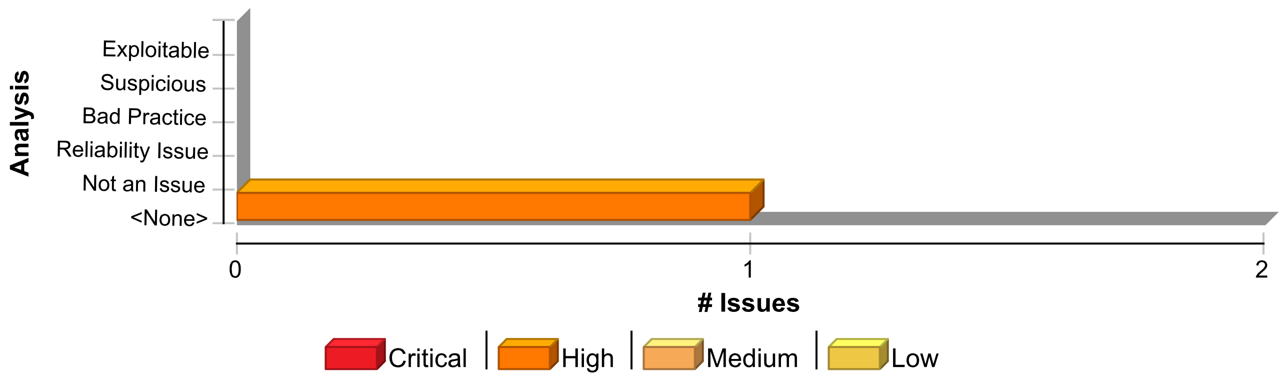
Some think that in the mobile world, classic vulnerabilities, such as path manipulation, do not make sense -- why would the user attack themself? However, keep in mind that the essence of mobile platforms is applications that are downloaded from various sources and run alongside each other on the same device. The likelihood of running a piece of malware next to a banking application is high, which necessitates expanding the attack surface of mobile applications to include inter-process communication. **Example 3:** The following code adapts `Example 1` to the Android platform.

```
...
        String rName = this.getIntent().getExtras().getString("reportName");
        File rFile = getBaseContext().getFileStreamPath(rName);
...
        rFile.delete();
...
```

Recommendation

The best way to prevent path manipulation is with a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. With this approach the input provided by the user is never used directly to specify the resource name. In some situations this approach is impractical because the set of legitimate resource names is too large or too hard to keep track of. Programmers often resort to blacklisting in these situations. Blacklisting selectively rejects or escapes potentially dangerous characters before using the input. However, any such list of unsafe characters is likely to be incomplete and will almost certainly become out of date. A better approach is to create a whitelist of characters that are allowed to appear in the resource name and accept input composed exclusively of characters in the approved set.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Path Manipulation	1	0	0	1
Total	**1**	**0**	**0**	**1**

Path Manipulation — High

Package: src

src/Perfil.java, line 46 (Path Manipulation) — High

Issue Details

Kingdom: Input Validation and Representation
Scan Engine: SCA (Data Flow)

Source Details

Source: Read java.lang.System.in
From: Main.main
File: src/Main.java:36

```
33  System.out.println(YELLOW + "###### ## ## ###### ## ## ## ########## ##
## ## ## ## ## " + RESET);
34
System.out.println("------------------------------------------------------------
35
System.out.println("------------------------------------------------------------
36  Scanner scanner = new Scanner(System.in);
37  boolean salir = false;
38  while (!salir) {
39  System.out.println(PURPLE + "----- MEN Ú PRINCIPAL -----" + RESET);
```

Sink Details

Sink: java.io.File.File()
Enclosing Method: mostrarContenidoPublicaciones()
File: src/Perfil.java:46
Taint Flags: STDIN

```
43  System.out.println("--------Publicaci n--------" + i);
44  // Reemplaza con la ruta de la foto que se muestra en la consola
```

Path Manipulation High

Package: src

src/Perfil.java, line 46 (Path Manipulation) High

```
String rutaFoto = publicacion.getRutaFoto();
File foto = new File(rutaFoto);
System.out.println(publicacion.toString());
Desktop.getDesktop().open(foto);
i++;
```

Poor Logging Practice: Use of a System Output Stream (58 issues)

Abstract

Using `System.out` or `System.err` rather than a dedicated logging facility makes it difficult to monitor the behavior of the program.

Explanation

Example 1: The first Java program that a developer learns to write is the following:

```
public class MyClass
  public static void main(String[] args) {
    System.out.println("hello world");
  }
}
```

While most programmers go on to learn many nuances and subtleties about Java, a surprising number hang on to this first lesson and never give up on writing messages to standard output using `System.out.println()`. The problem is that writing directly to standard output or standard error is often used as an unstructured form of logging. Structured logging facilities provide features like logging levels, uniform formatting, a logger identifier, timestamps, and, perhaps most critically, the ability to direct the log messages to the right place. When the use of system output streams is jumbled together with the code that uses loggers properly, the result is often a well-kept log that is missing critical information. Developers widely accept the need for structured logging, but many continue to use system output streams in their "pre-production" development. If the code you are reviewing is past the initial phases of development, use of `System.out` or `System.err` may indicate an oversight in the move to a structured logging system.

Recommendation

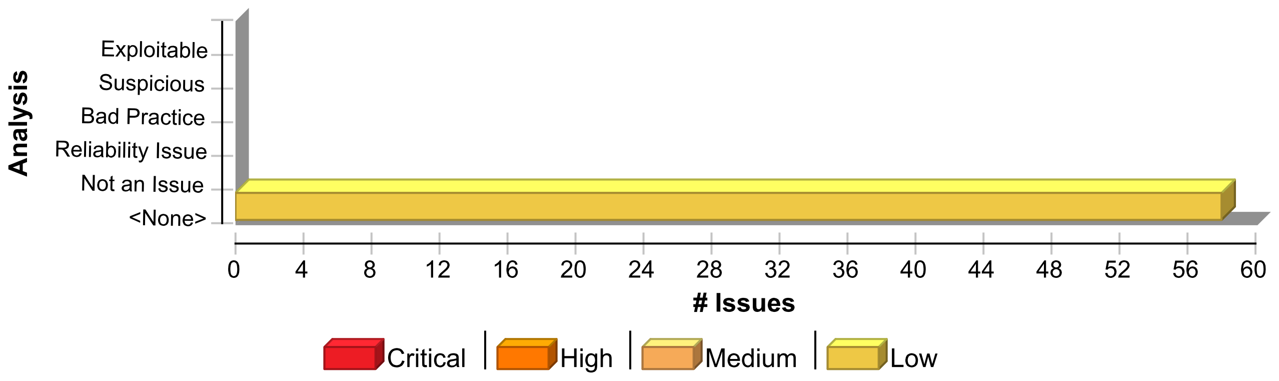
Use a Java logging facility rather than `System.out` or `System.err`. **Example 2:** For example, the "hello world" program in `Example 1` can be rewritten using log4j as follows:

```
import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;

public class MyClass {
  private final static Logger logger =
            Logger.getLogger(MyClass.class);

  public static void main(String[] args) {
    BasicConfigurator.configure();
    logger.info("hello world");
  }
}
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Poor Logging Practice: Use of a System Output Stream	58	0	0	58
Total	**58**	**0**	**0**	**58**

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Usuario.java, line 160 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: verPosibleUsuarios()
File: src/Usuario.java:160
Taint Flags:

```
157 System.out.println(contador + ". " + usuario.getNombre() + " (Mi usuario) ");
158 contador++;
159 } else {
160 System.out.println(contador + ". " + usuario.getNombre());
161 contador++;
162 }
163 }
```

src/Usuario.java, line 153 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Usuario.java, line 153 (Poor Logging Practice: Use of a System Output Stream) — Low

Enclosing Method: verPosibleUsuarios()
File: src/Usuario.java:153
Taint Flags:

```
150  public int verPosibleUsuarios() {
151  Scanner scanner = new Scanner(System.in);
152  int eleccion;
153  System.out.println("Usuarios disponibles:");
154  int contador = 1;
155  for (Usuario usuario : listaUsuarios) {
156  if (usuario == this) {
```

src/Usuario.java, line 79 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: print
Enclosing Method: agregarPublicacion()
File: src/Usuario.java:79
Taint Flags:

```
76  for (int i = 0; i < archivos.length; i++) {
77  System.out.println((i + 1) + ". " + archivos[i].getPath());
78  }
79  System.out.print("Seleccione el n mero de la imagen que desea publicar: ");
80  int opcion = scanner.nextInt();
81  scanner.nextLine(); // Limpiar el b fer
82  if (opcion < 1 || opcion > archivos.length) {
```

src/Usuario.java, line 111 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: seguirUsuario()
File: src/Usuario.java:111
Taint Flags:

```
108  int indice = verPosibleUsuarios();
109  Usuario usuarioSeleccionado = listaUsuarios.get(indice - 1);
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Usuario.java, line 111 (Poor Logging Practice: Use of a System Output Stream) — Low

```
if (usuarioSeleccionado == this) {
System.out.println("No puedes seguirte a ti mismo");
return;
}
Perfil perfil = usuarioSeleccionado.getPerfil();
```

src/Main.java, line 110 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: print
Enclosing Method: menuSecundario()
File: src/Main.java:110
Taint Flags:

```
System.out.println(WHITE + "3. Ver seguidores" + RESET);
System.out.println(PURPLE + "4. Seguir Usuarios" + RESET);
System.out.println(RED + "5. Salir" + RESET);
System.out.print("Ingrese una opci  n: ");
int opcion = scanner.nextInt();
scanner.nextLine(); // Limpiar el b  fer
switch (opcion) {
```

src/Perfil.java, line 97 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: agregarSeguidor()
File: src/Perfil.java:97
Taint Flags:

```
}
seguidores.add(seguidor);
incrementarSeguidores();
System.out.println("Has comenzado a seguir a " + seguidor.getNombre());
}

 /**
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 30 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:30
Taint Flags:

```
27  System.out.println(RED + "###### ## ## ###### ######## ## ########### ########## ## ####
####" + RESET);
28  System.out.println(PURPLE + " ## ### ## ## ## #### ## ## ## #### #### #### " + RESET);
29  System.out.println(YELLOW + " ## ## # ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
30  System.out.println(WHITE + " ## ## # ## ###### ## ## ## ## #### ## ##### ## ## ## ## ## "
+ RESET);
31  System.out.println(PURPLE + " ## ## ### ## ## ########## ## ## ## ## ########## ## ## " +
RESET);
32  System.out.println(RED + " ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
33  System.out.println(YELLOW + "###### ## ## ###### ## ## ## ########## ## ## ## ## ## ## " +
RESET);
```

src/Usuario.java, line 87 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: print
Enclosing Method: agregarPublicacion()
File: src/Usuario.java:87
Taint Flags:

```
84  return;
85  }
86  String rutaFoto = archivos[opcion - 1].getPath();
87  System.out.print("Ingrese su descripci n: ");
88  String descripcion = scanner.nextLine();
89  perfil.agregarPublicacion(new Publicacion(this, rutaFoto, descripcion));
90  System.out.println("-------Publicaci n realizada con  xito----");
```

src/Main.java, line 79 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 79 (Poor Logging Practice: Use of a System Output Stream) — Low

Sink Details

Sink: FunctionCall: println
Enclosing Method: cambiarUsuario()
File: src/Main.java:79
Taint Flags:

```
76  }
77
78  public static void cambiarUsuario(Scanner scanner) {
79  System.out.println("----- CAMBIAR DE USUARIO -----");
80  if (listaUsuarios.isEmpty()) {
81  System.out.println("No hay usuarios disponibles.");
82  return;
```

src/Main.java, line 29 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:29
Taint Flags:

```
26
System.out.println("----------------------------------------------------------------------------
27  System.out.println(RED + "###### ## ## ###### ######## ## ########### ########## ## ####
####" + RESET);
28  System.out.println(PURPLE + " ## ### ## ## ## #### ## ## ## #### #### #### " + RESET);
29  System.out.println(YELLOW + " ## ## # ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
30  System.out.println(WHITE + " ## ## # ## ###### ## ## ## ## #### ## ##### ## ## ## ## ## "
+ RESET);
31  System.out.println(PURPLE + " ## ## ### ## ## ########## ## ## ## ## ########## ## ## " +
RESET);
32  System.out.println(RED + " ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
```

src/Main.java, line 84 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: cambiarUsuario()
File: src/Main.java:84

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 84 (Poor Logging Practice: Use of a System Output Stream) — Low

Taint Flags:

```
System.out.println("No hay usuarios disponibles.");
return;
}
System.out.println("Usuarios disponibles:");
for (int i = 0; i < listaUsuarios.size(); i++) {
Usuario usuario = listaUsuarios.get(i);
System.out.println((i + 1) + ". " + usuario.getNombre());
```

src/Usuario.java, line 164 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: verPosibleUsuarios()
File: src/Usuario.java:164
Taint Flags:

```
contador++;
}
}
System.out.println("Seleccione uno:");
int i = 0;
eleccion = scanner.nextInt();
do {
```

src/Main.java, line 104 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: menuSecundario()
File: src/Main.java:104
Taint Flags:

```
public static void menuSecundario(Scanner scanner) throws IOException {
boolean salir = false;
while (!salir) {
System.out.println(PURPLE + "----- MEN ̗ SECUNDARIO -----" + RESET);
System.out.println(RED + "1. Agregar publicaci  n" + RESET);
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 104 (Poor Logging Practice: Use of a System Output Stream) — Low

```
106  System.out.println(YELLOW + "2. Ver perfil" + RESET);
107  System.out.println(WHITE + "3. Ver seguidores" + RESET);
```

src/Usuario.java, line 83 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: agregarPublicacion()
File: src/Usuario.java:83
Taint Flags:

```
80  int opcion = scanner.nextInt();
81  scanner.nextLine(); // Limpiar el b  fer
82  if (opcion < 1 || opcion > archivos.length) {
83  System.out.println("Opci  n inv  lida. No se realiz  la publicaci  n.");
84  return;
85  }
86  String rutaFoto = archivos[opcion - 1].getPath();
```

src/Main.java, line 105 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: menuSecundario()
File: src/Main.java:105
Taint Flags:

```
102  boolean salir = false;
103  while (!salir) {
104  System.out.println(PURPLE + "----- MEN⊬ SECUNDARIO -----" + RESET);
105  System.out.println(RED + "1. Agregar publicaci  n" + RESET);
106  System.out.println(YELLOW + "2. Ver perfil" + RESET);
107  System.out.println(WHITE + "3. Ver seguidores" + RESET);
108  System.out.println(PURPLE + "4. Seguir Usuarios" + RESET);
```

src/Usuario.java, line 157 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Usuario.java, line 157 (Poor Logging Practice: Use of a System Output Stream) — Low

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: verPosibleUsuarios()
File: src/Usuario.java:157
Taint Flags:

```
154  int contador = 1;
155  for (Usuario usuario : listaUsuarios) {
156  if (usuario == this) {
157  System.out.println(contador + ". " + usuario.getNombre() + " (Mi usuario) ");
158  contador++;
159  } else {
160  System.out.println(contador + ". " + usuario.getNombre());
```

src/Main.java, line 33 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:33
Taint Flags:

```
30  System.out.println(WHITE + " ## ## # ## ###### ## ## ## ## #### ## ##### ## ## ## ## ## "
+ RESET);
31  System.out.println(PURPLE + " ## ## ### ## ## ########## ## ## ## ## ########## ## ## " +
RESET);
32  System.out.println(RED + " ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
33  System.out.println(YELLOW + "###### ## ## ###### ## ## ## ########## ## ## ## ## ## ## " +
RESET);
34
System.out.println("---------------------------------------------------------------------------
35
System.out.println("---------------------------------------------------------------------------
36  Scanner scanner = new Scanner(System.in);
```

src/Main.java, line 109 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 109 (Poor Logging Practice: Use of a System Output Stream) — Low

Sink Details

Sink: FunctionCall: println
Enclosing Method: menuSecundario()
File: src/Main.java:109
Taint Flags:

```
106  System.out.println(YELLOW + "2. Ver perfil" + RESET);
107  System.out.println(WHITE + "3. Ver seguidores" + RESET);
108  System.out.println(PURPLE + "4. Seguir Usuarios" + RESET);
109  System.out.println(RED + "5. Salir" + RESET);
110  System.out.print("Ingrese una opci  n: ");
111  int opcion = scanner.nextInt();
112  scanner.nextLine(); // Limpiar el b  fer
```

src/Main.java, line 107 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: menuSecundario()
File: src/Main.java:107
Taint Flags:

```
104  System.out.println(PURPLE + "----- MEN Ú SECUNDARIO -----" + RESET);
105  System.out.println(RED + "1. Agregar publicaci  n" + RESET);
106  System.out.println(YELLOW + "2. Ver perfil" + RESET);
107  System.out.println(WHITE + "3. Ver seguidores" + RESET);
108  System.out.println(PURPLE + "4. Seguir Usuarios" + RESET);
109  System.out.println(RED + "5. Salir" + RESET);
110  System.out.print("Ingrese una opci  n: ");
```

src/Perfil.java, line 43 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: mostrarContenidoPublicaciones()
File: src/Perfil.java:43
Taint Flags:

```
40  public void mostrarContenidoPublicaciones() throws IOException {
41  int i = 1;
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Perfil.java, line 43 (Poor Logging Practice: Use of a System Output Stream) — Low

```
for (Publicacion publicacion : publicaciones) {
System.out.println("--------Publicaci n--------" + i);
// Reemplaza con la ruta de la foto que se muestra en la consola
String rutaFoto = publicacion.getRutaFoto();
File foto = new File(rutaFoto);
```

src/Perfil.java, line 47 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: mostrarContenidoPublicaciones()
File: src/Perfil.java:47
Taint Flags:

```
// Reemplaza con la ruta de la foto que se muestra en la consola
String rutaFoto = publicacion.getRutaFoto();
File foto = new File(rutaFoto);
System.out.println(publicacion.toString());
Desktop.getDesktop().open(foto);
i++;
}
```

src/Main.java, line 130 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: menuSecundario()
File: src/Main.java:130
Taint Flags:

```
salir = true;
break;
default:
System.out.println(RED + "Opci n inv lida. Intente nuevamente." + RESET);
break;
}
}
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 39 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:39
Taint Flags:

```
36  Scanner scanner = new Scanner(System.in);
37  boolean salir = false;
38  while (!salir) {
39  System.out.println(PURPLE + "----- MEN ↓ PRINCIPAL -----" + RESET);
40  System.out.println(RED + "1. Agregar nuevo usuario" + RESET);
41  System.out.println(YELLOW + "2. Elegir usuario" + RESET);
42  System.out.println(WHITE + "3. Salir" + RESET);
```

src/Main.java, line 32 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:32
Taint Flags:

```
29  System.out.println(YELLOW + " ## ## # ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
30  System.out.println(WHITE + " ## ## # ## ###### ## ## ## ## #### ## ##### ## ## ## ## ## "
+ RESET);
31  System.out.println(PURPLE + " ## ## ### ## ## ########## ## ## ## ## ########## ## ## " +
RESET);
32  System.out.println(RED + " ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
33  System.out.println(YELLOW + "###### ## ## ###### ## ## ## ########## ## ## ## ## ## ## " +
RESET);
34
System.out.println("-----------------------------------------------------------------------
35
System.out.println("-----------------------------------------------------------------------
```

src/Perfil.java, line 91 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Perfil.java, line 91 (Poor Logging Practice: Use of a System Output Stream) — Low

Sink Details

Sink: FunctionCall: println
Enclosing Method: agregarSeguidor()
File: src/Perfil.java:91
Taint Flags:

```
88 public void agregarSeguidor(Usuario seguidor) {
89 for (Usuario seguidores : seguidores) {
90 if (seguidores.equals(seguidor)) {
91 System.out.println("Ya sigues a " + seguidor.getNombre());
92 return;
93 }
94 }
```

src/Main.java, line 108 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: menuSecundario()
File: src/Main.java:108
Taint Flags:

```
105 System.out.println(RED + "1. Agregar publicaci  n" + RESET);
106 System.out.println(YELLOW + "2. Ver perfil" + RESET);
107 System.out.println(WHITE + "3. Ver seguidores" + RESET);
108 System.out.println(PURPLE + "4. Seguir Usuarios" + RESET);
109 System.out.println(RED + "5. Salir" + RESET);
110 System.out.print("Ingrese una opci  n: ");
111 int opcion = scanner.nextInt();
```

src/Main.java, line 89 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: print
Enclosing Method: cambiarUsuario()
File: src/Main.java:89
Taint Flags:

```
86 Usuario usuario = listaUsuarios.get(i);
87 System.out.println((i + 1) + ". " + usuario.getNombre());
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 89 (Poor Logging Practice: Use of a System Output Stream) — Low

```
}
System.out.print("Ingrese el n  mero de usuario al que desea cambiar: ");
int opcion = scanner.nextInt();
scanner.nextLine(); // Limpiar el b  fer
if (opcion < 1 || opcion > listaUsuarios.size()) {
```

src/Usuario.java, line 77 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: agregarPublicacion()
File: src/Usuario.java:77
Taint Flags:

```
}
System.out.println("Rutas de im  genes disponibles:");
for (int i = 0; i < archivos.length; i++) {
System.out.println((i + 1) + ". " + archivos[i].getPath());
}
System.out.print("Seleccione el n  mero de la imagen que desea publicar: ");
int opcion = scanner.nextInt();
```

src/Main.java, line 43 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: print
Enclosing Method: main()
File: src/Main.java:43
Taint Flags:

```
System.out.println(RED + "1. Agregar nuevo usuario" + RESET);
System.out.println(YELLOW + "2. Elegir usuario" + RESET);
System.out.println(WHITE + "3. Salir" + RESET);
System.out.print("Ingrese una opci  n: ");
int opcion = scanner.nextInt();
scanner.nextLine(); // Limpiar el b  fer
switch (opcion) {
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Usuario.java, line 128 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: verPerfil()
File: src/Usuario.java:128
Taint Flags:

```
125  int eleccion = this.verPosibleUsuarios();
126  Usuario usuarioSeleccionado = listaUsuarios.get(eleccion - 1);
127  if (!(usuarioSeleccionado == this)) {
128  System.out.println(usuarioSeleccionado.perfil.toString());
129  usuarioSeleccionado.verPublicaciones();
130  return usuarioSeleccionado;
131  }
```

src/Main.java, line 28 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:28
Taint Flags:

```
25  // Crear usuarios
26
System.out.println("----------------------------------------------------------------------------
27  System.out.println(RED + "###### ## ## ###### ######## ## ########### ########## ## ####
####" + RESET);
28  System.out.println(PURPLE + " ## ### ## ## ## #### ## ## ## #### #### #### " + RESET);
29  System.out.println(YELLOW + " ## ## # ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
30  System.out.println(WHITE + " ## ## # ## ###### ## ## ## ## #### ## ##### ## ## ## ## ## "
+ RESET);
31  System.out.println(PURPLE + " ## ## ### ## ## ########## ## ## ## ## ########## ## ## " +
RESET);
```

src/Main.java, line 106 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 106 (Poor Logging Practice: Use of a System Output Stream) — Low

Sink Details

Sink: FunctionCall: println
Enclosing Method: menuSecundario()
File: src/Main.java:106
Taint Flags:

```
103  while (!salir) {
104  System.out.println(PURPLE + "----- MEN↓ SECUNDARIO -----" + RESET);
105  System.out.println(RED + "1. Agregar publicaci  n" + RESET);
106  System.out.println(YELLOW + "2. Ver perfil" + RESET);
107  System.out.println(WHITE + "3. Ver seguidores" + RESET);
108  System.out.println(PURPLE + "4. Seguir Usuarios" + RESET);
109  System.out.println(RED + "5. Salir" + RESET);
```

src/Main.java, line 60 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:60
Taint Flags:

```
57  salir = true;
58  break;
59  default:
60  System.out.println(RED + "Opci  n inv  lida. Intente nuevamente." + RESET);
61  break;
62  }
63  }
```

src/Usuario.java, line 169 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: verPosibleUsuarios()
File: src/Usuario.java:169
Taint Flags:

```
166  eleccion = scanner.nextInt();
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Usuario.java, line 169 (Poor Logging Practice: Use of a System Output Stream) — Low

```
do {
if (!(eleccion >= 1 && eleccion <= listaUsuarios.size())) {
System.out.println("Eleccion no valida, ingrese de nuevo");
}
scanner.nextLine();
i++;
```

src/Main.java, line 87 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: cambiarUsuario()
File: src/Main.java:87
Taint Flags:

```
System.out.println("Usuarios disponibles:");
for (int i = 0; i < listaUsuarios.size(); i++) {
Usuario usuario = listaUsuarios.get(i);
System.out.println((i + 1) + ". " + usuario.getNombre());
}
System.out.print("Ingrese el n mero de usuario al que desea cambiar: ");
int opcion = scanner.nextInt();
```

src/Usuario.java, line 72 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: agregarPublicacion()
File: src/Usuario.java:72
Taint Flags:

```
// Directorio de im genes
"C:\\Images");
if (archivos.length == 0) {
System.out.println("No se encontraron im genes en el directorio especificado.");
return;
}
System.out.println("Rutas de im genes disponibles:");
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Usuario.java, line 72 (Poor Logging Practice: Use of a System Output Stream) — Low

src/Perfil.java, line 105 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: mostrarSeguidores()
File: src/Perfil.java:105
Taint Flags:

```
102  * @param seguidores La lista de seguidores a mostrar.
103  */
104  public void mostrarSeguidores(List<Usuario> seguidores) {
105  System.out.println("-----SEGUIDORES-----");
106  for (int i = 0; i < seguidores.size(); i++) {
107  Usuario seguidor = seguidores.get(i);
108  System.out.println("Seguidor " + (i + 1) + ": " + seguidor.getNombre());
```

src/Main.java, line 31 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:31
Taint Flags:

```
28  System.out.println(PURPLE + " ## ### ## ## ## #### ## ## ## #### #### #### " + RESET);
29  System.out.println(YELLOW + " ## ## # ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
30  System.out.println(WHITE + " ## ## # ## ###### ## ## ## ## #### ## ##### ## ## ## ## ## "
+ RESET);
31  System.out.println(PURPLE + " ## ## ### ## ## ########## ## ## ## ## ########## ## ## " +
RESET);
32  System.out.println(RED + " ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
33  System.out.println(YELLOW + "###### ## ## ###### ## ## ## ########## ## ## ## ## ## ## " +
RESET);
34
System.out.println("---------------------------------------------------------------------
```

src/Main.java, line 64 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 64 (Poor Logging Practice: Use of a System Output Stream) — Low

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:64
Taint Flags:

```
61  break;
62  }
63  }
64  System.out.println("  Hasta luego!");
65  }
66
67  public static void crearUsuario(Scanner scanner) {
```

src/Main.java, line 26 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:26
Taint Flags:

```
23
24  public static void main(String[] args) throws IOException {
25  // Crear usuarios
26
System.out.println("---------------------------------------------------------------------------
27  System.out.println(RED + "###### ## ## ###### ######## ## ########### ########## ## ####
####" + RESET);
28  System.out.println(PURPLE + " ## ### ## ## ## #### ## ## ## #### #### #### " + RESET);
29  System.out.println(YELLOW + " ## ## # ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
```

src/Main.java, line 34 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 34 (Poor Logging Practice: Use of a System Output Stream) — Low

Enclosing Method: main()
File: src/Main.java:34
Taint Flags:

```
31  System.out.println(PURPLE + " ## ## ### ## ## ########## ## ## ## ## ########## ## ## " +
RESET);
32  System.out.println(RED + " ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
33  System.out.println(YELLOW + "###### ## ## ###### ## ## ## ########## ## ## ## ## ## ## " +
RESET);
34
System.out.println("--------------------------------------------------------------------------
35
System.out.println("--------------------------------------------------------------------------
36  Scanner scanner = new Scanner(System.in);
37  boolean salir = false;
```

src/Main.java, line 35 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:35
Taint Flags:

```
32  System.out.println(RED + " ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
33  System.out.println(YELLOW + "###### ## ## ###### ## ## ## ########## ## ## ## ## ## ## " +
RESET);
34
System.out.println("--------------------------------------------------------------------------
35
System.out.println("--------------------------------------------------------------------------
36  Scanner scanner = new Scanner(System.in);
37  boolean salir = false;
38  while (!salir) {
```

src/Main.java, line 93 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: cambiarUsuario()
File: src/Main.java:93

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 93 (Poor Logging Practice: Use of a System Output Stream) — Low

Taint Flags:

```
90  int opcion = scanner.nextInt();
91  scanner.nextLine(); // Limpiar el b  fer
92  if (opcion < 1 || opcion > listaUsuarios.size()) {
93  System.out.println("Opci  n inv  lida. No se realiz  el cambio de usuario.");
94  usuarioActual = null;
95  } else {
96  usuarioActual = listaUsuarios.get(opcion - 1);
```

src/Main.java, line 41 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:41
Taint Flags:

```
38  while (!salir) {
39  System.out.println(PURPLE + "----- MEN ⇂ PRINCIPAL -----" + RESET);
40  System.out.println(RED + "1. Agregar nuevo usuario" + RESET);
41  System.out.println(YELLOW + "2. Elegir usuario" + RESET);
42  System.out.println(WHITE + "3. Salir" + RESET);
43  System.out.print("Ingrese una opci  n: ");
44  int opcion = scanner.nextInt();
```

src/Main.java, line 97 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: cambiarUsuario()
File: src/Main.java:97
Taint Flags:

```
94  usuarioActual = null;
95  } else {
96  usuarioActual = listaUsuarios.get(opcion - 1);
97  System.out.println("  Bienvenido, " + usuarioActual.getNombre() + "!");
98  }
99  }
```

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 97 (Poor Logging Practice: Use of a System Output Stream) — Low

```
100
```

src/Usuario.java, line 132 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: verPerfil()
File: src/Usuario.java:132
Taint Flags:

```
usuarioSeleccionado.verPublicaciones();
return usuarioSeleccionado;
}
System.out.println(this.perfil.toString());
this.verPublicaciones();
return this;
}
```

src/Main.java, line 75 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: crearUsuario()
File: src/Main.java:75
Taint Flags:

```
String contrasena = scanner.nextLine();
Usuario usuario = new Usuario(nombreUsuario, contrasena);
listaUsuarios.add(usuario);
System.out.println("Usuario creado exitosamente.");
}

public static void cambiarUsuario(Scanner scanner) {
```

src/Main.java, line 81 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 81 (Poor Logging Practice: Use of a System Output Stream) — Low

Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: cambiarUsuario()
File: src/Main.java:81
Taint Flags:

```
78  public static void cambiarUsuario(Scanner scanner) {
79  System.out.println("----- CAMBIAR DE USUARIO -----");
80  if (listaUsuarios.isEmpty()) {
81  System.out.println("No hay usuarios disponibles.");
82  return;
83  }
84  System.out.println("Usuarios disponibles:");
```

src/Main.java, line 42 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:42
Taint Flags:

```
39  System.out.println(PURPLE + "----- MEN ⇂ PRINCIPAL -----" + RESET);
40  System.out.println(RED + "1. Agregar nuevo usuario" + RESET);
41  System.out.println(YELLOW + "2. Elegir usuario" + RESET);
42  System.out.println(WHITE + "3. Salir" + RESET);
43  System.out.print("Ingrese una opci  n: ");
44  int opcion = scanner.nextInt();
45  scanner.nextLine(); // Limpiar el b  fer
```

src/Main.java, line 27 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:27
Taint Flags:

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Main.java, line 27 (Poor Logging Practice: Use of a System Output Stream) — Low

```
24  public static void main(String[] args) throws IOException {
25  // Crear usuarios
26
System.out.println("------------------------------------------------------------------------------------
27  System.out.println(RED + "###### ## ## ###### ######## ## ########### ########## ## ####
####" + RESET);
28  System.out.println(PURPLE + " ## ### ## ## ## #### ## ## ## #### #### #### " + RESET);
29  System.out.println(YELLOW + " ## ## # ## ## ## ## ## ## ## ## ## ## ## ## ## ## " + RESET);
30  System.out.println(WHITE + " ## ## # ## ###### ## ## ## ## #### ## ##### ## ## ## ## ## "
+ RESET);
```

src/Main.java, line 71 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: print
Enclosing Method: crearUsuario()
File: src/Main.java:71
Taint Flags:

```
68  System.out.println("----- CREAR USUARIO -----");
69  System.out.print("Ingrese el nombre de usuario: ");
70  String nombreUsuario = scanner.nextLine();
71  System.out.print("Ingrese la contrase  a: ");
72  String contrasena = scanner.nextLine();
73  Usuario usuario = new Usuario(nombreUsuario, contrasena);
74  listaUsuarios.add(usuario);
```

src/Main.java, line 69 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: print
Enclosing Method: crearUsuario()
File: src/Main.java:69
Taint Flags:

```
66
67  public static void crearUsuario(Scanner scanner) {
68  System.out.println("----- CREAR USUARIO -----");
69  System.out.print("Ingrese el nombre de usuario: ");
70  String nombreUsuario = scanner.nextLine();
```

Poor Logging Practice: Use of a System Output Stream

Low

Package: src

src/Main.java, line 69 (Poor Logging Practice: Use of a System Output Stream)

Low

```
71  System.out.print("Ingrese la contrase  a: ");
72  String contrasena = scanner.nextLine();
```

src/Usuario.java, line 107 (Poor Logging Practice: Use of a System Output Stream)

Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: seguirUsuario()
File: src/Usuario.java:107
Taint Flags:

```
104  * Permite al usuario seguir a otro usuario.
105  */
106  public void seguirUsuario() {
107  System.out.println("-----USUARIOS DISPONIBLES PARA SEGUIR-----");
108  int indice = verPosibleUsuarios();
109  Usuario usuarioSeleccionado = listaUsuarios.get(indice - 1);
110  if (usuarioSeleccionado == this) {
```

src/Usuario.java, line 90 (Poor Logging Practice: Use of a System Output Stream)

Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: agregarPublicacion()
File: src/Usuario.java:90
Taint Flags:

```
87  System.out.print("Ingrese su descripci  n: ");
88  String descripcion = scanner.nextLine();
89  perfil.agregarPublicacion(new Publicacion(this, rutaFoto, descripcion));
90  System.out.println("-------Publicaci  n realizada con   xito----");
91  this.verPublicaciones();
92  }
93
```

src/Usuario.java, line 75 (Poor Logging Practice: Use of a System Output Stream)

Low

Issue Details

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Usuario.java, line 75 (Poor Logging Practice: Use of a System Output Stream) — Low

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: agregarPublicacion()
File: src/Usuario.java:75
Taint Flags:

```
72  System.out.println("No se encontraron im genes en el directorio especificado.");
73  return;
74  }
75  System.out.println("Rutas de im genes disponibles:");
76  for (int i = 0; i < archivos.length; i++) {
77  System.out.println((i + 1) + ". " + archivos[i].getPath());
78  }
```

src/Main.java, line 68 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: crearUsuario()
File: src/Main.java:68
Taint Flags:

```
65  }
66
67  public static void crearUsuario(Scanner scanner) {
68  System.out.println("----- CREAR USUARIO -----");
69  System.out.print("Ingrese el nombre de usuario: ");
70  String nombreUsuario = scanner.nextLine();
71  System.out.print("Ingrese la contrase a: ");
```

src/Perfil.java, line 108 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: mostrarSeguidores()
File: src/Perfil.java:108

Poor Logging Practice: Use of a System Output Stream — Low

Package: src

src/Perfil.java, line 108 (Poor Logging Practice: Use of a System Output Stream) — Low

Taint Flags:

```
105  System.out.println("-----SEGUIDORES-----");
106  for (int i = 0; i < seguidores.size(); i++) {
107  Usuario seguidor = seguidores.get(i);
108  System.out.println("Seguidor " + (i + 1) + ": " + seguidor.getNombre());
109  }
110  }
111
```

src/Main.java, line 40 (Poor Logging Practice: Use of a System Output Stream) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: println
Enclosing Method: main()
File: src/Main.java:40
Taint Flags:

```
37  boolean salir = false;
38  while (!salir) {
39  System.out.println(PURPLE + "----- MEN Ɩ PRINCIPAL -----" + RESET);
40  System.out.println(RED + "1. Agregar nuevo usuario" + RESET);
41  System.out.println(YELLOW + "2. Elegir usuario" + RESET);
42  System.out.println(WHITE + "3. Salir" + RESET);
43  System.out.print("Ingrese una opci  n: ");
```

Poor Style: Identifier Contains Dollar Symbol ($) (1 issue)

Abstract

Using a dollar sign ($) as part of an identifier is not recommended.

Explanation

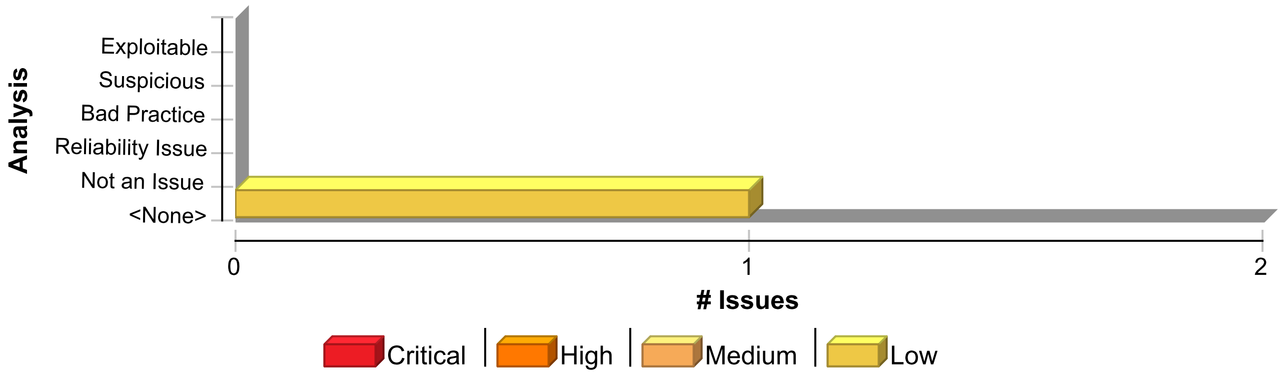
Section 3.8 of the Java Language Specification reserves the dollar sign ($) for identifiers that are used only in mechanically generated source code. **Example:**

```
int un$afe;
```

Recommendation

Rename identifiers that use the dollar sign ($) to names that do not carry an overloaded meaning.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Poor Style: Identifier Contains Dollar Symbol ($)	1	0	0	1
Total	**1**	**0**	**0**	**1**

Poor Style: Identifier Contains Dollar Symbol ($) **Low**

Package: src

src/Usuario.java, line 27 (Poor Style: Identifier Contains Dollar Symbol ($)) **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Variable: $missing$
Enclosing Method: Usuario()
File: src/Usuario.java:27
Taint Flags:

```
24  * @param nombre el nombre del usuario
```

Poor Style: Identifier Contains Dollar Symbol ($) **Low**

Package: src

src/Usuario.java, line 27 (Poor Style: Identifier Contains Dollar Symbol ($)) **Low**

```
25  * @param contrase  a la contrase  a del usuario
26  */
27  public Usuario(String nombre, String contrase  a) {
28  this.nombre = nombre;
29  this.contrase  a = contrase  a;
30  this.perfil = new Perfil(this);
```

Poor Style: Value Never Read (1 issue)

Abstract

The variable's value is assigned but never used, making it a dead store.

Explanation

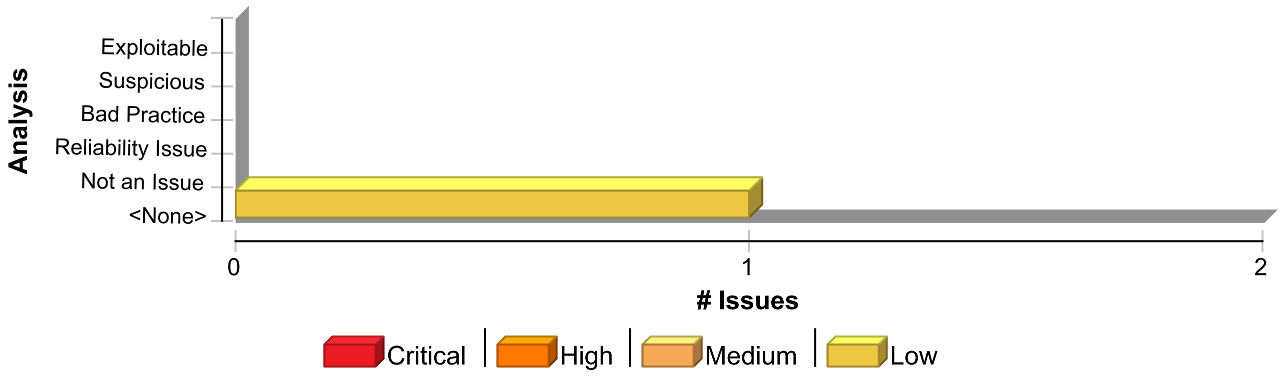
This variable's value is not used. After the assignment, the variable is either assigned another value or goes out of scope. **Example:** The following code excerpt assigns to the variable `r` and then overwrites the value without using it.

```
  r = getName();
  r = getNewBuffer(buf);
```

Recommendation

Remove unnecessary assignments in order to make the code easier to understand and maintain.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Poor Style: Value Never Read	1	0	0	1
Total	**1**	**0**	**0**	**1**

Poor Style: Value Never Read — **Low**

Package: src

src/Usuario.java, line 172 (Poor Style: Value Never Read) — **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: VariableAccess: i
Enclosing Method: verPosibleUsuarios()
File: src/Usuario.java:172
Taint Flags:

Poor Style: Value Never Read | Low

Package: src

src/Usuario.java, line 172 (Poor Style: Value Never Read) | Low

```
169  System.out.println("Eleccion no valida, ingrese de nuevo");
170  }
171  scanner.nextLine();
172  i++;
173  return eleccion;
174  } while (i != 1);
175  }
```

Portability Flaw: File Separator (1 issue)

Abstract

The use of hardcoded file separators causes portability problems.

Explanation

Different operating systems use different characters as file separators. For example, Microsoft Windows systems use "\", while UNIX systems use "/". When applications have to run on different platforms, the use of hardcoded file separators can lead to incorrect execution of application logic and potentially a denial of service. **Example 1:** The following code uses a hardcoded file separator to open a file:

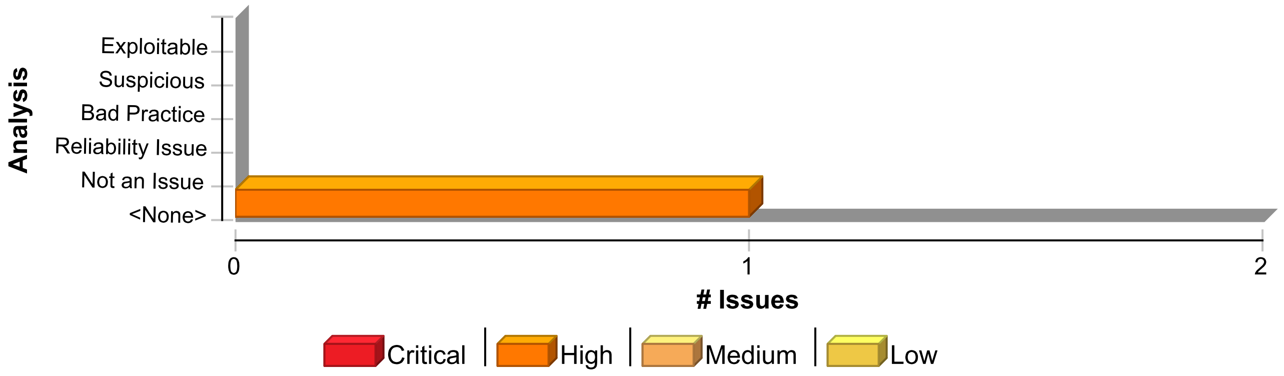
```
...
File file = new File(directoryName + "\\" + fileName);
...
```

Recommendation

In order to write portable code, avoid using hardcoded file separators. Instead, rely on platform-independent APIs provided by the language library. **Example 2:** The following code implements the same functionality as `Example 1`, but uses platform-independent API to specify a file separator:

```
...
File file = new File(directoryName + File.separator + fileName);
...
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Portability Flaw: File Separator	1	0	0	1
Total	**1**	**0**	**0**	**1**

Portability Flaw: File Separator — **High**

Package: src

src/Usuario.java, line 184 (Portability Flaw: File Separator) — **High**

Issue Details

Portability Flaw: File Separator — High

Package: src

src/Usuario.java, line 184 (Portability Flaw: File Separator) — High

Kingdom: Code Quality
Scan Engine: SCA (Data Flow)

Source Details

Source: Read
From: Usuario.agregarPublicacion
File: src/Usuario.java:68

```
65  */
66  public void agregarPublicacion() throws IOException {
67  Scanner scanner = new Scanner(System.in);
68  File[] archivos = obtenerArchivosDirectorio(
69  // Directorio de im genes
70  "C:\\Images");
71  if (archivos.length == 0) {
```

Sink Details

Sink: java.io.File.File()
Enclosing Method: obtenerArchivosDirectorio()
File: src/Usuario.java:184
Taint Flags: FILESEPARATOR

```
181  * @return los archivos de imagen en el directorio
182  */
183  private File[] obtenerArchivosDirectorio(String directorio) {
184  File folder = new File(directorio);
185  return folder.listFiles((dir, name) -> name.toLowerCase().endsWith(".jpeg")
186  || name.toLowerCase().endsWith(".jpg") || name.toLowerCase().endsWith(".png"));
187  }
```

Portability Flaw: Locale Dependent Comparison (3 issues)

Abstract

Unexpected portability problems can be found when the locale is not specified.

Explanation

When comparing data that may be locale-dependent, an appropriate locale should be specified. **Example 1:** The following example tries to perform validation to determine if user input includes a

Recommendation

To prevent this from occurring, always make sure to either specify the default locale, or specify the locale with APIs that accept them such as `toUpperCase()`. **Example 2:** The following specifies the locale manually as an argument to `toUpperCase()`.

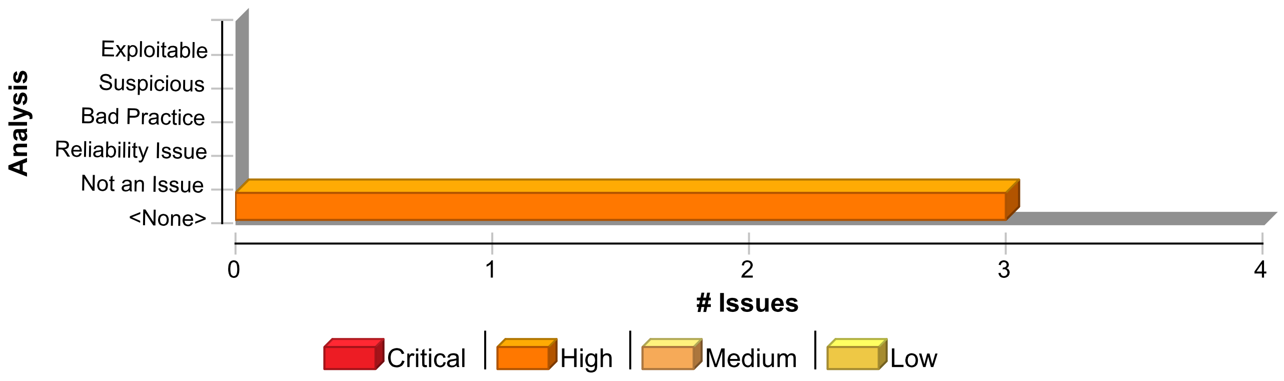
```
import java.util.Locale;
  ...
  public String tagProcessor(String tag){
    if (tag.toUpperCase(Locale.ENGLISH).equals("SCRIPT")){
      return null;
    }
    //does not contain SCRIPT tag, keep processing input
    ...
  }
  ...
```

Example 3: The following uses the function `java.lang.String.equalsIgnoreCase()` API to prevent this issue.

```
  ...
  public String tagProcessor(String tag){
    if (tag.equalsIgnoreCase("SCRIPT")){
      return null;
    }
    //does not contain SCRIPT tag, keep processing input
    ...
  }
  ...
```

This prevents the problem because `equalsIgnoreCase()` changes case similar to `Character.toLowerCase()` and `Character.toUpperCase()`. This involves creating temporary canonical forms of both strings using information from the `UnicodeData` file that is part of the Unicode Character Database maintained by the Unicode Consortium, and even though this may render them unreadable if they were to be read out, it makes comparison possible without being dependent upon locale.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Portability Flaw: Locale Dependent Comparison	3	0	0	3
Total	**3**	**0**	**0**	**3**

Portability Flaw: Locale Dependent Comparison — High

Package: src

src/Usuario.java, line 186 (Portability Flaw: Locale Dependent Comparison) — High

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: name.toLowerCase().endsWith(...) : Comparison without checking locale
Enclosing Method: accept()
File: src/Usuario.java:186
Taint Flags:

```
183  private File[] obtenerArchivosDirectorio(String directorio) {
184  File folder = new File(directorio);
185  return folder.listFiles((dir, name) -> name.toLowerCase().endsWith(".jpeg")
186  || name.toLowerCase().endsWith(".jpg") || name.toLowerCase().endsWith(".png"));
187  }
188
189  /**
```

src/Usuario.java, line 185 (Portability Flaw: Locale Dependent Comparison) — High

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: name.toLowerCase().endsWith(...) : Comparison without checking locale
Enclosing Method: accept()
File: src/Usuario.java:185
Taint Flags:

Portability Flaw: Locale Dependent Comparison — High

Package: src

src/Usuario.java, line 185 (Portability Flaw: Locale Dependent Comparison) — High

```
182  */
183  private File[] obtenerArchivosDirectorio(String directorio) {
184  File folder = new File(directorio);
185  return folder.listFiles((dir, name) -> name.toLowerCase().endsWith(".jpeg")
186  || name.toLowerCase().endsWith(".jpg") || name.toLowerCase().endsWith(".png"));
187  }
188
```

src/Usuario.java, line 186 (Portability Flaw: Locale Dependent Comparison) — High

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: name.toLowerCase().endsWith(...) : Comparison without checking locale
Enclosing Method: accept()
File: src/Usuario.java:186
Taint Flags:

```
183  private File[] obtenerArchivosDirectorio(String directorio) {
184  File folder = new File(directorio);
185  return folder.listFiles((dir, name) -> name.toLowerCase().endsWith(".jpeg")
186  || name.toLowerCase().endsWith(".jpg") || name.toLowerCase().endsWith(".png"));
187  }
188
189  /**
```

System Information Leak: Internal (2 issues)

Abstract

Revealing system data or debugging information helps an adversary learn about the system and form a plan of attack.

Explanation

An internal information leak occurs when system data or debugging information is sent to a local file, console, or screen via printing or logging. **Example 1:** The following code writes an exception to the standard error stream:

```
try {
    ...
} catch (Exception e) {
    e.printStackTrace();
}
```

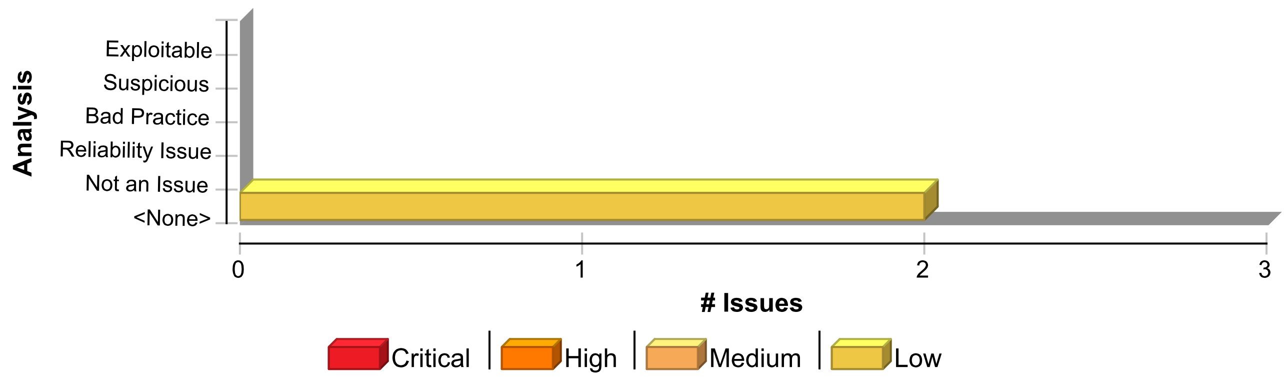
Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a user. In some cases the error message tells the attacker precisely what sort of an attack the system is vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In `Example 1`, the leaked information could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program. In the mobile world, information leaks are also a concern. **Example 2:** The following code logs the stack trace of a caught exception on the Android platform.

```
...
try {
  ...
} catch (Exception e) {
    Log.e(TAG, Log.getStackTraceString(e));
}
...
```

Recommendation

Write error messages with security in mind. In production environments, turn off detailed error information in favor of brief messages. Restrict the generation and storage of detailed output that can help administrators and programmers diagnose problems. Be careful, debugging traces can sometimes appear in non-obvious places (embedded in comments in the HTML for an error page, for example). Even brief error messages that do not reveal stack traces or database dumps can potentially aid an attacker. For example, an "Access Denied" message can reveal that a file or user exists on the system.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
System Information Leak: Internal	2	0	0	2
Total	**2**	**0**	**0**	**2**

System Information Leak: Internal — Low

Package: src

src/Perfil.java, line 47 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.io.File.listFiles()
From: Usuario.obtenerArchivosDirectorio
File: src/Usuario.java:185

```
182  */
183  private File[] obtenerArchivosDirectorio(String directorio) {
184  File folder = new File(directorio);
185  return folder.listFiles((dir, name) ->
name.toLowerCase().endsWith(".jpeg")
186  || name.toLowerCase().endsWith(".jpg") ||
name.toLowerCase().endsWith(".png"));
187  }
188
```

Sink Details

Sink: java.io.PrintStream.println()
Enclosing Method: mostrarContenidoPublicaciones()
File: src/Perfil.java:47
Taint Flags: SYSTEMINFO

```
44  // Reemplaza con la ruta de la foto que se muestra en la consola
45  String rutaFoto = publicacion.getRutaFoto();
46  File foto = new File(rutaFoto);
```

System Information Leak: Internal — Low

Package: src

src/Perfil.java, line 47 (System Information Leak: Internal) — Low

```
System.out.println(publicacion.toString());
Desktop.getDesktop().open(foto);
i++;
}
```

src/Usuario.java, line 77 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.io.File.listFiles()
From: Usuario.obtenerArchivosDirectorio
File: src/Usuario.java:185

```
*/
private File[] obtenerArchivosDirectorio(String directorio) {
File folder = new File(directorio);
return folder.listFiles((dir, name) ->
name.toLowerCase().endsWith(".jpeg")
|| name.toLowerCase().endsWith(".jpg") ||
name.toLowerCase().endsWith(".png"));
}

```

Sink Details

Sink: java.io.PrintStream.println()
Enclosing Method: agregarPublicacion()
File: src/Usuario.java:77
Taint Flags: SYSTEMINFO

```
}
System.out.println("Rutas de im genes disponibles:");
for (int i = 0; i < archivos.length; i++) {
System.out.println((i + 1) + ". " + archivos[i].getPath());
}
System.out.print("Seleccione el n mero de la imagen que desea publicar: ");
int opcion = scanner.nextInt();
```