

## **INFORME DE AUDITORÍA DE SEGURIDAD MÓVIL**

Evaluación de vulnerabilidades en la aplicación AllSafe

Elaborado por:

- Artem
- Nikolai
- Lucas
- Anthony

Ciclo Formativo de Desarrollo de Aplicaciones Multiplataforma (D.A.M.)

Centro de Formación Profesional Txurdinaga

2025

## **Índice**

- 1. Introducción**
- 2. Metodología**
- 3. Vulnerabilidades Analizadas**
  - 3.1 Hardcoded Credentials**
  - 3.2 Insecure Logging**
  - 3.3 Insecure Shared Preferences**
  - 3.4 Insecure Broadcast Receiver**
  - 3.5 SQL Injection**
  - 3.6 Certificate Pinning**
- 4. Tabla Comparativa de Vulnerabilidades**
- 5. Conclusiones Finales**

## **1. Introducción**

El presente informe recoge los resultados de una auditoría de seguridad móvil realizada sobre la aplicación AllSafe, desarrollada con fines formativos para identificar y corregir vulnerabilidades comunes en entornos Android.

El objetivo es reconocer vulnerabilidades, evaluar su severidad e impacto y proponer medidas de mitigación siguiendo las directrices del OWASP Mobile Security Testing Guide (MSTG).

## **2. Metodología**

La auditoría se llevó a cabo mediante ingeniería inversa, análisis estático y dinámico de la aplicación. Se emplearon las siguientes herramientas:

- JADX para la descompilación del código fuente.
- ADB y logcat para la monitorización del comportamiento en ejecución.
- BurpSuite para el análisis de tráfico HTTP/HTTPS.
- Frida para inyección de scripts y bypass de protecciones.

El entorno de pruebas se configuró en un emulador Android con privilegios root.



Revisar las constantes/variables definidas en la clase y el contenido de BODY (SOAP payload).

Localizar UsernameToken / PasswordText.



### Detección del cuerpo SOAP (credenciales incrustadas)

Durante la revisión de la clase se observó una variable constante llamada BODY que contiene un **payload SOAP** en formato XML.

```
public static final String BODY =  
    "<soap:Envelope xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">\n" +  
    "<soap:Header>\n" +  
    "<UsernameToken xmlns=\"http://siebel.com/webservices/\">\n" +  
    "<Username>superadmin</Username>\n" +  
    "<PasswordText>supersecurepassword</PasswordText>\n" +  
    "</UsernameToken>\n" +  
    "</soap:Header>\n";
```

Al examinar el código fuente, se hace evidente que hay otra variable declarada como cadena, que recupera su valor usando getString(R.string.dev\_env).

```

intrinsic.checkNotNull(expression.value(view.findViewById(...));
Button request = (Button) view.findViewById(
23 request.setOnClickListener(new View.OnClickListener() { // from class: infosecadventures.allsafe.challenges.HardcodedCredentials$$ExternalSynth
@Override // method from View.OnClickListener
public final void onClick(View view2) {
HardcodedCredentials.onCreateView$lambda$0(this, view2);
}
});
42 return view;
}

/* JADX INFO: Access modifiers changed from: private */
23 public static final void onCreateView$lambda$0(HardcodedCredentials this$0, View it) {
24 OkHttpClient client = new OkHttpClient();
25 RequestBody body = RequestBody.INSTANCE.create(BODY, SOAP);
26 Request.Builder builder = new Request.Builder();
27 String string = this$0.getString(R.string.dev_env);
28 Intrinsic.checkNotNull(expression.value(string), "getString(...));
29 Request req = builder.url(string).post(body).build();
30 client.newCall(req).enqueue(new Callback() { // from class: infosecadventures.allsafe.challenges.HardcodedCredentials$onCreateView$1$1
@Override // okhttp3.Callback
33 public void onResponse(Call call, Response response) {
32 Intrinsic.checkNotNullParameter(call, "call");
Intrinsic.checkNotNullParameter(response, "response");
}
}

@Override // okhttp3.Callback
37 public void onFailure(Call call, IOException e) {
Intrinsic.checkNotNullParameter(call, "call");
Intrinsic.checkNotNullParameter(e, "e");
}
}

```

Esto indica la presencia de una variable definida en el archivo strings.xml ubicado en res/values/strings.xml, llamada dev\_env.

Abrir Resources → res/values/strings.xml y buscar dev\_env.

```

> kotlin
> kotlinox
> okhttp3
> okio
> org
> Recursos
> assets
> kotlin
> lib
> META-INF
> okhttp3
> res
71 <string name="dev_env">https://admin:password123@dev.infosecadventures.com</string>
72 <string name="error_ally_label">Error: invalid</string>
73 <string name="error_icon_content_description">Error</string>
74 <string name="exposed_dropdown_menu_content_description">Show dropdown menu</string>
75 <string name="fab_transformation_scrim_behavior">com.google.android.material.transformation.FabTransformationScrimBehavior</string>
76 <string name="fab_transformation_sheet_behavior">com.google.android.material.transformation.FabTransformationSheetBehavior</string>
77 <string name="fallback_menu_item_copy_link">Copy link</string>
78 <string name="fallback_menu_item_open_in_browser">Open in browser</string>
79 <string name="fallback_menu_item_share_link">Share link</string>
80 <string name="firebase_database_url">https://allsafe-8cef0.firebaseio.com</string>
81 <string name="acm_defaultSenderId">983632168629</string>

```

## Identificación del recurso getString(R.string.dev\_env)

En Android, el patrón getString(R.string.algo) significa que la aplicación está leyendo un valor del archivo de recursos (res/values/strings.xml).

R.string.dev\_env no es una variable local ni constante; es un **identificador de recurso** generado automáticamente durante la compilación.

Al inspeccionar el árbol de recursos en JADX (Resources → res → values → strings.xml), se encontró la siguiente entrada:

```
<string name="dev_env">https://admin:password123@dev.infosecadventures.com</string>
```

## Mitigación

- Eliminar cualquier credencial incrustada dentro del código fuente o archivos de recursos (strings.xml).
- Mover las credenciales a un **servidor backend seguro** y obtener los tokens de acceso en tiempo de ejecución mediante una API autenticada.
- 
- Usar **EncryptedSharedPreferences** para almacenar **valores generados dinámicamente** (como tokens de sesión o configuraciones temporales).

## Severidad

**Alta (High)** — la exposición de credenciales en texto plano permite a un atacante autenticarse directamente en servicios internos (dev.infosecadventures.com), comprometiendo el entorno de desarrollo o producción.

Este tipo de vulnerabilidad se clasifica como **“Insecure Storage of Sensitive Information” (MSTG-STORAGE-2)** según el OWASP Mobile Security Testing Guide y requiere corrección inmediata.

## Conclusión de hallazgo

Se identificaron **dos fuentes de credenciales hardcodeadas**:

Estas credenciales pueden ser extraídas fácilmente mediante ingeniería inversa de la APK, lo que



viola las buenas prácticas del OWASP Mobile Security Testing Guide (MSTG-STORAGE-2) al exponer datos sensibles en texto plano.

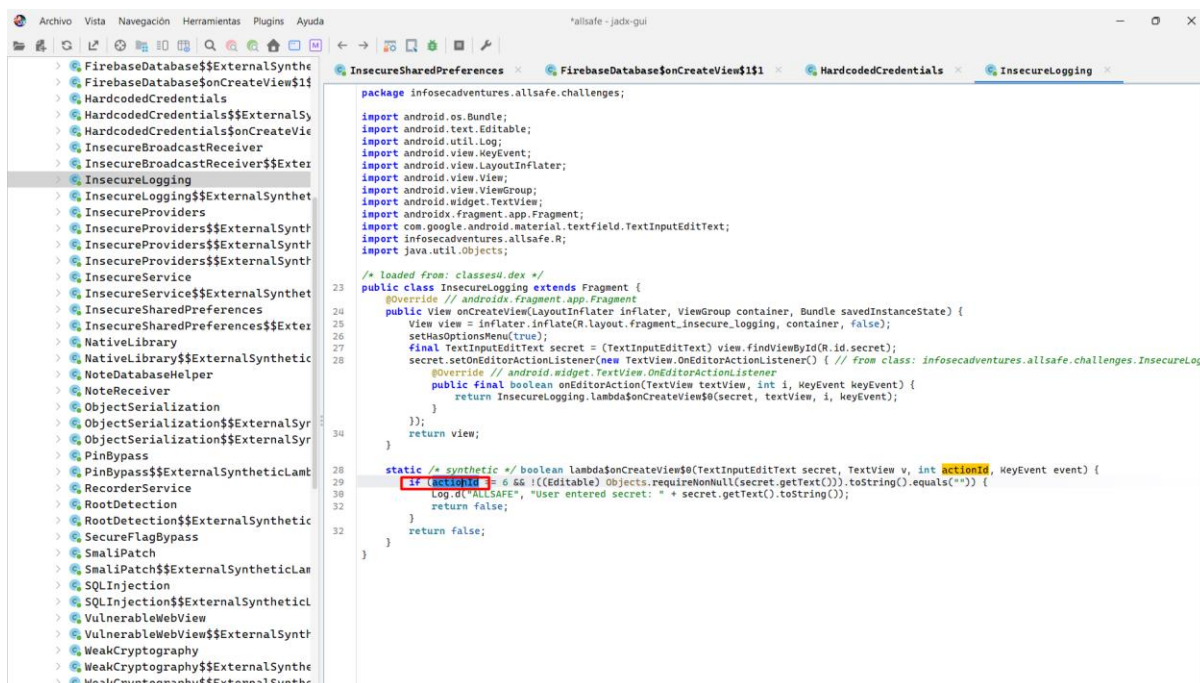
Esta evidencia demuestra el almacenamiento inseguro de credenciales en la aplicación, accesibles mediante ingeniería inversa, lo que podría permitir acceso no autorizado a servicios internos o pruebas de autenticación.

### 3.2 Insecure Logging

#### Desarrollo

Esta aplicación registra la cadena ingresada por el usuario en los registros de la aplicación de forma insegura. Este registro solo ocurre si el usuario hace clic en el botón “Listo” en el teclado de Android. Si utiliza un emulador y presiona la tecla “Enter”.

Primero analizaremos en JADX, el fichero InsecureLogging



Al descompilar la clase `infosecadventures.allsafe.challenges.InsecureLogging` se ve un `secret.OnEditorActionListener` que hace el log solo cuando `actionId == 6` (que corresponde a `IME_ACTION_DONE` — el botón “Done” del teclado virtual). Por eso debes pulsar ese botón para que se ejecute el log.

```
Log.d("ALLSAFE", "User entered secret: " + secret.getText().toString());
```

**Tag fijo:** "ALLSAFE".

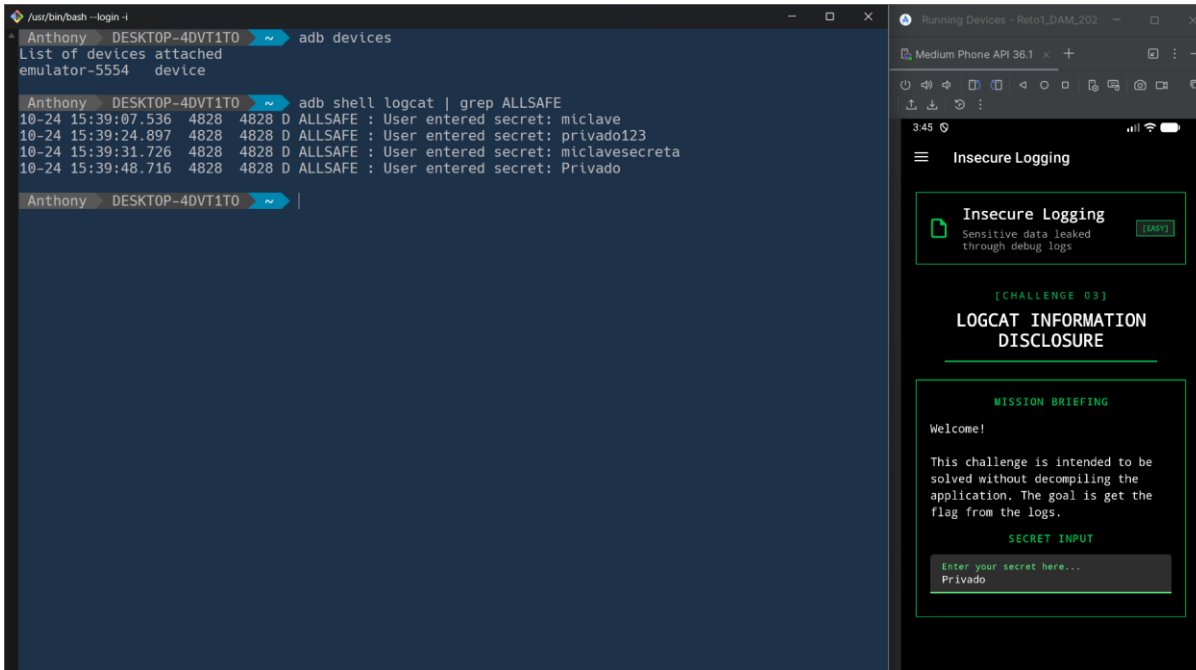
**Mensaje:** concatena el **texto introducido por el usuario**  
(`secret.getText().toString()`).

**Nivel:** `Log.d` (debug) → saldrá en `logcat`.

Ahora consultaremos el `logcat` filtrando por la palabra `ALLSAFE`, para obtener las claves secretas

Comando:

```
adb shell logcat | grep ALLSAFE
```



### Severidad

#### Alta.

El registro de datos sensibles (como contraseñas o “secrets”) mediante `Log.d()` en tiempo de ejecución puede exponer información crítica a cualquier usuario o aplicación con acceso al sistema de logs. En un dispositivo rooteado o con apps con permisos de depuración, estos datos pueden ser fácilmente extraídos.

### Mitigación

- Eliminar cualquier uso de `Log.d`, `System.out.println`, `Timber.d` u otros métodos de logging que muestren datos sensibles.
- Si los logs son necesarios para depuración, envolverlos con una condición:

```
if (BuildConfig.DEBUG) {  
  
    Log.d("TAG", "Mensaje de depuración seguro");  
  
}
```

}

Desactivar `android:debuggable="true"` en el `AndroidManifest.xml` para entornos de producción.

Aplicar una **política de logging seguro**, donde nunca se registren contraseñas, tokens o PII.

## Conclusión

Durante el análisis del componente `InsecureLogging`, se comprobó que la aplicación registra en texto plano el valor introducido por el usuario bajo el tag `ALLSAFE`.

Al ejecutar `adb shell logcat | grep ALLSAFE`, se evidenció la exposición del dato sensible en los logs del sistema.

Este comportamiento constituye una **vulnerabilidad de seguridad**, ya que permite la fuga de información a través de registros accesibles. La implementación debe corregirse para garantizar la confidencialidad de los datos del usuario.

## Tarea Certificate Pinning

Para esta tarea vamos a configurar el emulador para usar el proxy de BurpSuite y añadir el CA de Burp Suite en emulador:

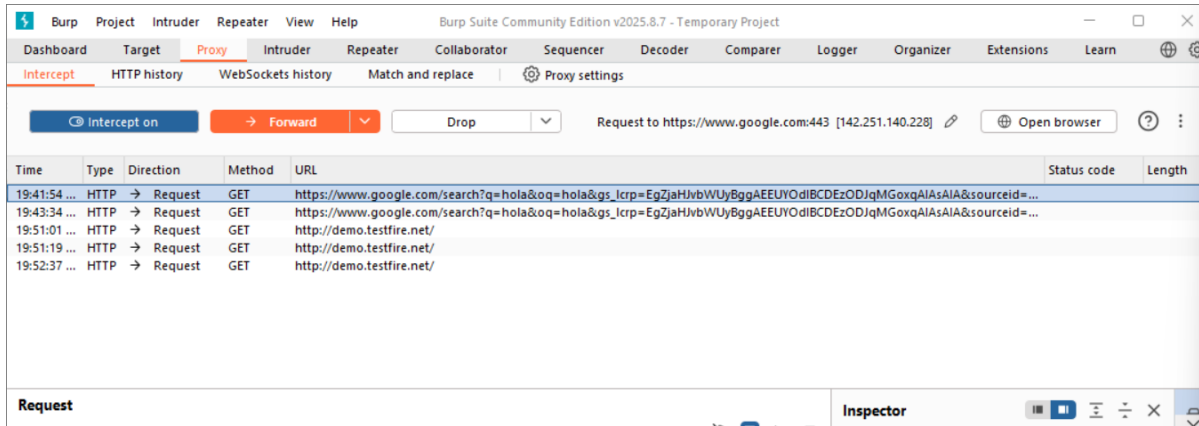
Primero vamos a cambiar el proxy del emulador para que BurpSuite pueda interceptar las peticiones, utilizaremos el `http_proxy 10.0.2.2:8080` de nuestra pc.

```
Cargar los perfiles personales y de sistema tardó 2835ms.
> adb devices
List of devices attached
emulator-5554    device

> adb shell
emu64xa:/ $ settings put global http_proxy 10.0.2.2:8080
emu64xa:/ $
```

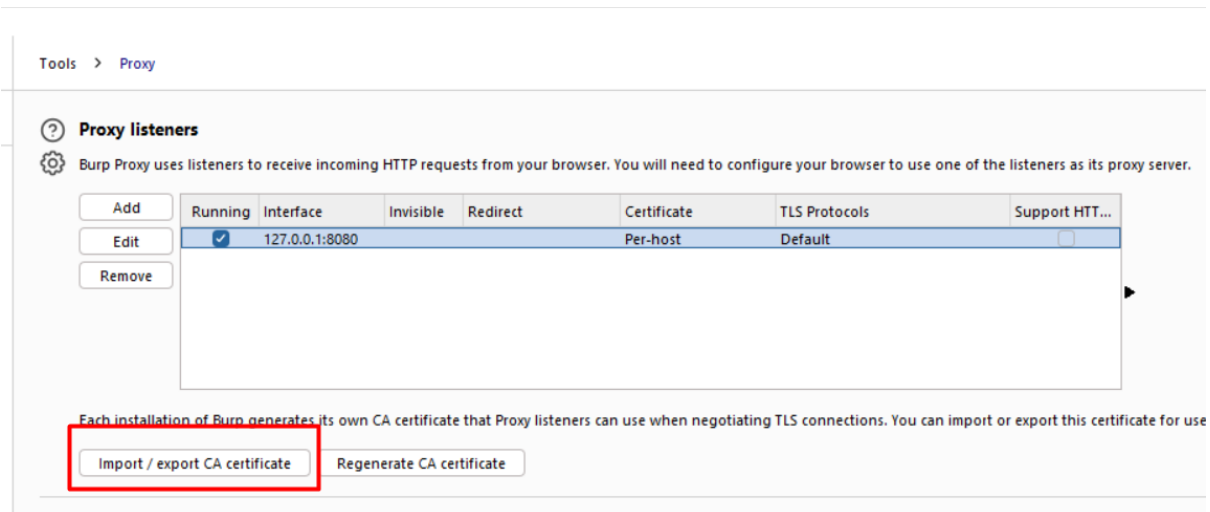
Luego verificamos que ya BurpSuite intercepta las peticiones.

## Auditoría de Seguridad Móvil – Aplicación AllSafe

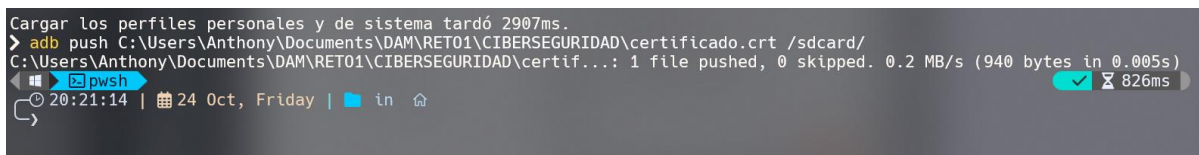


Seguido de esto vamos a exportar e instalar el CA de BurpSuite en nuestro emulador.

Primero vamos a exportar el certificado.



Luego instalaremos el CA en nuestro emulador.



Para instalar el certificado obtenido por BurpSuite dentro del emulador

vamos a Ajustes > Seguridad & Privacidad > Mas ajustes y privacidad > Encriptación & Credenciales > Instalar un certificado > Certificado CA

Aqui buscamos el certificado que subimos al emulador lo seleccionamos y lo instalamos.

Una vez ya instalado, ejecutaremos frida-ps en nuestro terminal y buscaremos el identificador de la app que en este caso es `infosecadventures.allsafe`

```
> frida-ps -aIU
PID  Name                                     Identifier
-----
4828  Allsafe                                infosecadventures.allsafe
10522 AndroGoat - Insecure App (Kotlin)      owasp.sat.agoat
8103  Chrome                                com.android.chrome
6964  Clock                                 com.google.android.deskclock
10796 Drive                               com.google.android.apps.docs
10739 Files                             com.google.android.documentsui
7547  Google                                com.google.android.googlequicksearchbox
5206  Google Play Store                     com.android.vending
7282  Messages                             com.google.android.apps.messaging
7073  Personal Safety                      com.google.android.apps.safetyhub
6820  Phone                                com.google.android.dialer
6405  Photos                               com.google.android.apps.photos
1272  SIM Toolkit                           com.android.stk
2922  Settings                             com.android.settings
-    Calendar                           com.google.android.calendar
-    Camera                              com.android.camera2
-    Contacts                           com.google.android.contacts
-    Gmail                              com.google.android.gm
-    Maps                               com.google.android.apps.maps
-    Reto1_DAM_2025-26                 com.example.reto1_dam_2025_26
-    Voice Access                      com.google.android.apps.accessibility.voiceaccess
-    YouTube                           com.google.android.youtube
-    YouTube Music                     com.google.android.apps.youtube.music
```

Luego subimos el servidor de frida al emulador.

```
> adb push C:\Users\Anthony\Documents\DAM\RET01\CIBERSEGURIDAD\frida-server-17.4.1-android-x86_64.xz /data/local/tmp/frida-server
C:\Users\Anthony\Documents\DAM\RET01\CIBERSEGURIDAD\frida-server-17.4.1-android-x86_64.xz: 1 file pushed, 0 skipped. 227.4 MB/s (31488632 bytes in 0.132s)
```

En otra terminal comprobamos que tenemos root y le damos permisos de su, y le daremos permisos al fichero que subimos. Para poder arrancar el servidor.

```

cargar los perfiles personales y de sistema cuando lo vamos a
> adb devices
List of devices attached
emulator-5554    device

> adb root
adb is already running as root
> adb shell
emu64xa:/ # su
emu64xa:/ # cd /data/local/tmp/
emu64xa:/data/local/tmp # # chmod +x frida-server-17.4.1-android-x86_64
emu64xa:/data/local/tmp # chmod +x frida-server-17.4.1-android-x86_64
emu64xa:/data/local/tmp # ./frida-server-17.4.1-android-x86_64

```

Ahora vamos a subir un script para realizar el Bypass, lo obtenemos de la plataforma de frida

<https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/>

```

pwsh
frida -U -r infosecadventures.allsafe -l C:\Users\Anthony\Documents\DAM\RET01\CIBERSEGURIDAD\sslbypass.js

Frida 17.4.1 - A world-class dynamic instrumentation toolkit

Commands:
  help      -> Displays the help system
  object?   -> Display information about 'object'
  exit/quit -> Exit

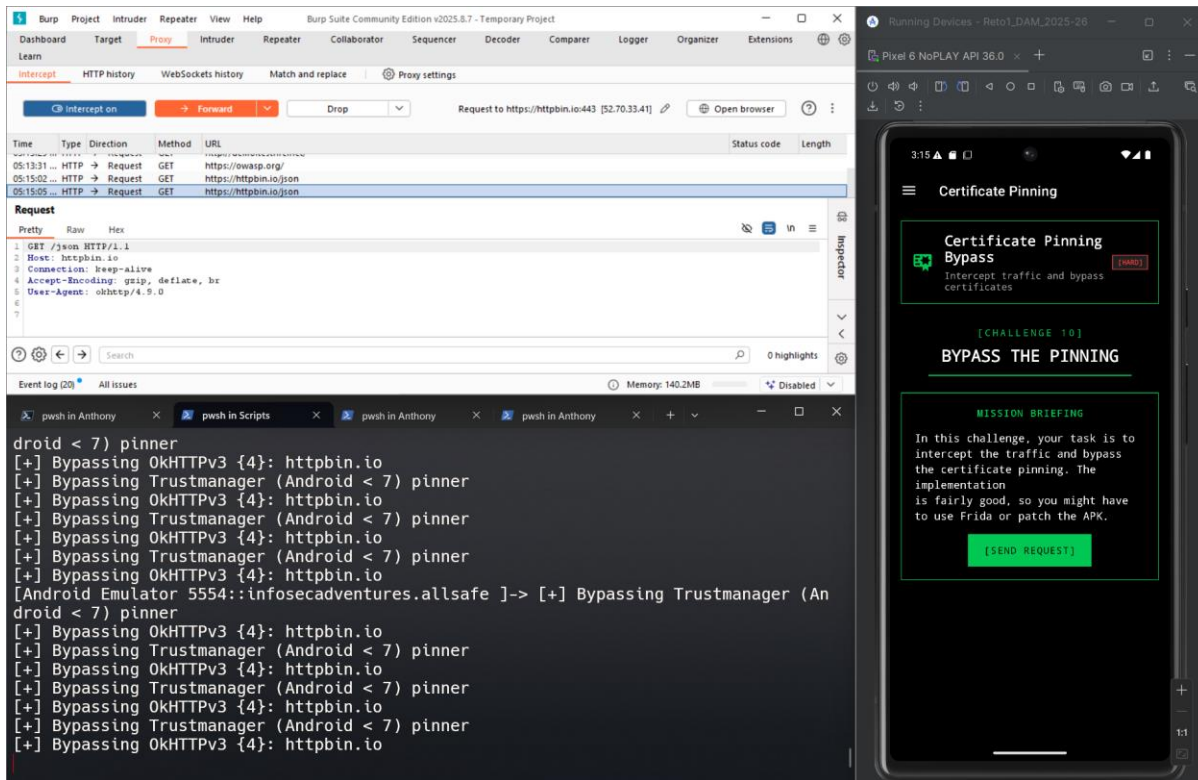
More info at https://frida.re/docs/home/

Connected to Android Emulator 5554 (id=emulator-5554)
Spawned 'infosecadventures.allsafe'. Resuming main thread!
[Android Emulator 5554:infosecadventures.allsafe ]->
=====
[#] Android Bypass for various Certificate Pinning methods [#]
=====
[-] OkHttpV3 {2} pinner not found
[-] Trustkit {1} pinner not found
[-] Trustkit {2} pinner not found
[-] Trustkit {3} pinner not found
[-] Appcelerator PinningTrustManager pinner not found
[-] Fabric PinningTrustManager pinner not found
[-] OpenSSLSocketImpl Conscrypt {1} pinner not found
[-] OpenSSLSocketImpl Conscrypt {2} pinner not found
[-] OpenSSLEngineSocketImpl Conscrypt pinner not found
[-] OpenSSLSocketImpl Apache Harmony pinner not found
[-] PhoneGap sslCertificateChecker pinner not found
[-] IBM MobileFirst pinTrustedCertificatePublicKey {1} pinner not found
[-] IBM MobileFirst pinTrustedCertificatePublicKey {2} pinner not found
[-] IBM WorkLight HostNameVerifierWithCertificatePinning {1} pinner not found
[-] IBM WorkLight HostNameVerifierWithCertificatePinning {2} pinner not found
[-] IBM WorkLight HostNameVerifierWithCertificatePinning {3} pinner not found
[-] IBM WorkLight HostNameVerifierWithCertificatePinning {4} pinner not found

```

Una vez que hemos ejecutado el script nos dirigimos al emulador y comprobamos en la sección de Certificate Pinning, presionamos Send Request, y comprobamos que el script está ejecutando correctamente y hemos realizado un bypass del certificado de BurpSuite con Frida.





### Severidad

**Alta.** La capacidad de interceptar o modificar tráfico TLS (ya sea porque no existe pinning o porque puede ser desactivado en tiempo de ejecución con Frida) compromete confidencialidad e integridad de la comunicación entre la app y el servidor. Esto permite exfiltración de credenciales, tokens, datos personales y manipulación de respuestas del backend. Si el pinning es inexistente o incorrectamente implementado (p. ej. desactivable en builds debug o por hooking), el riesgo es crítico para cualquier API sensible.

### Mitigación

1. **Implementar pinning de clave pública (public-key pinning)** en la capa de red (ej. `okhttp3.CertificatePinner`) con *SHA-256* de la clave pública, no pins de certificado completos. Incluir al menos una *backup pin* para rotaciones seguras.



## 2. Proteger contra manipulación en tiempo de ejecución:

- Detectar entornos con root/jailbreak y comportamientos de depuración (rechazar o restringir funcionalidades).
- Integridad del binario (checksum, verificación de firma/Signature) y evadir modificaciones (anti-tamper).
- Evitar habilitar bypass en builds **DEBUG**; separar configuración y no incluir “switches” desactivables por variables accesibles.

## 3. Ofuscación y monitoreo: ofuscar implementación y métodos críticos (R8/ProGuard), aplicar detección de instrumentation (Frida, Xposed) y alertar/registrar intentos de manipulación.

## Conclusión

Durante la prueba se configuró el emulador para usar Burp, se instaló la CA de Burp y, además, se empleó Frida para inyectar un script de bypass; tras esto se consiguió interceptar/forzar tráfico HTTPS. Esto muestra que actualmente la aplicación no protege de forma efectiva la capa TLS frente a ataques de man-in-the-middle cuando el runtime puede ser manipulado — ya sea por ausencia de pinning o por una implementación que puede ser desactivada en ejecución.

# Insecure Broadcast Receiver

primero entramos en nuestra terminal

ahi llegamos hasta el archivo donde tenemos adb en mi caso es

```
Directorio de C:\Users\2153760H\Documents\platform-tools-latest-windows\platform-tools
```

comprobamos que el emulador esta conectado con adb devices

```
C:\Users\2153760H\Documents\platform-tools-latest-windows\platform-tools>adb devices
List of devices attached
emulator-5554    device
```

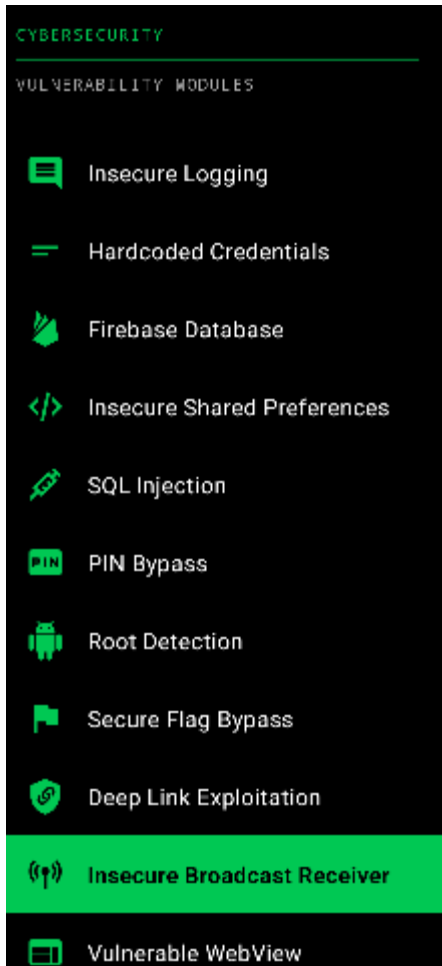
limpiamos los logcat po si acaso con comando adb logcat -c

```
C:\Users\2153760H\Documents\platform-tools-latest-windows\platform-tools>adb logcat -c
```

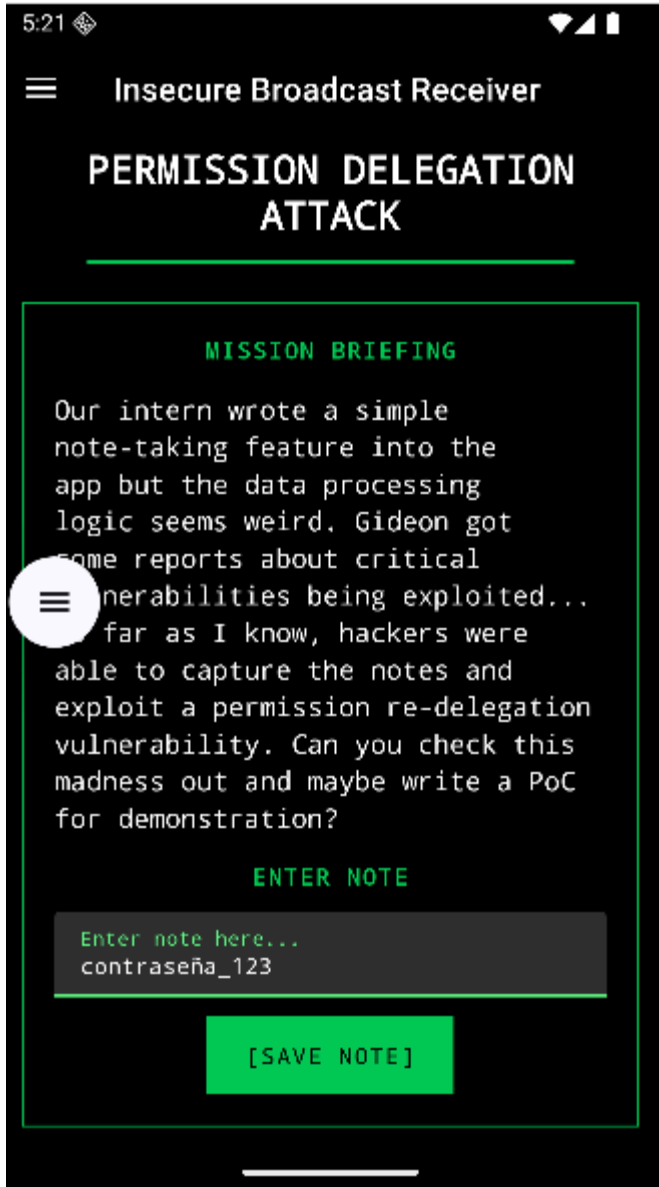
ahora ejecutamos monitoreo ESPECÍFICO con comando adb logcat | findstr -i "allsafe"

```
C:\Users\2153760H\Documents\platform-tools-latest-windows\platform-tools>adb logcat | findstr -i "allsafe"
```

Ahora entramos en nuestro emulador y ejecutamos allsafe, ahi vamos a Insecure Broadcast Reseiver



y ahí vas a ver un campo para escribir texto, ahí escribe lo que sea, en mi caso es contraseña\_123



y dele a save y el terminal va a ver algo como 10-23 17:10:50.279 6218 6218 D ALLSAFE:  
[http://prod.allsafe.infosecadventures.io/api/v1/note/add?auth\\_token=YWxsc2FmZV9kZXZfYWRTaW5fdG9rZW4%3D&note=contrase%C3%B1a\\_123](http://prod.allsafe.infosecadventures.io/api/v1/note/add?auth_token=YWxsc2FmZV9kZXZfYWRTaW5fdG9rZW4%3D&note=contrase%C3%B1a_123)

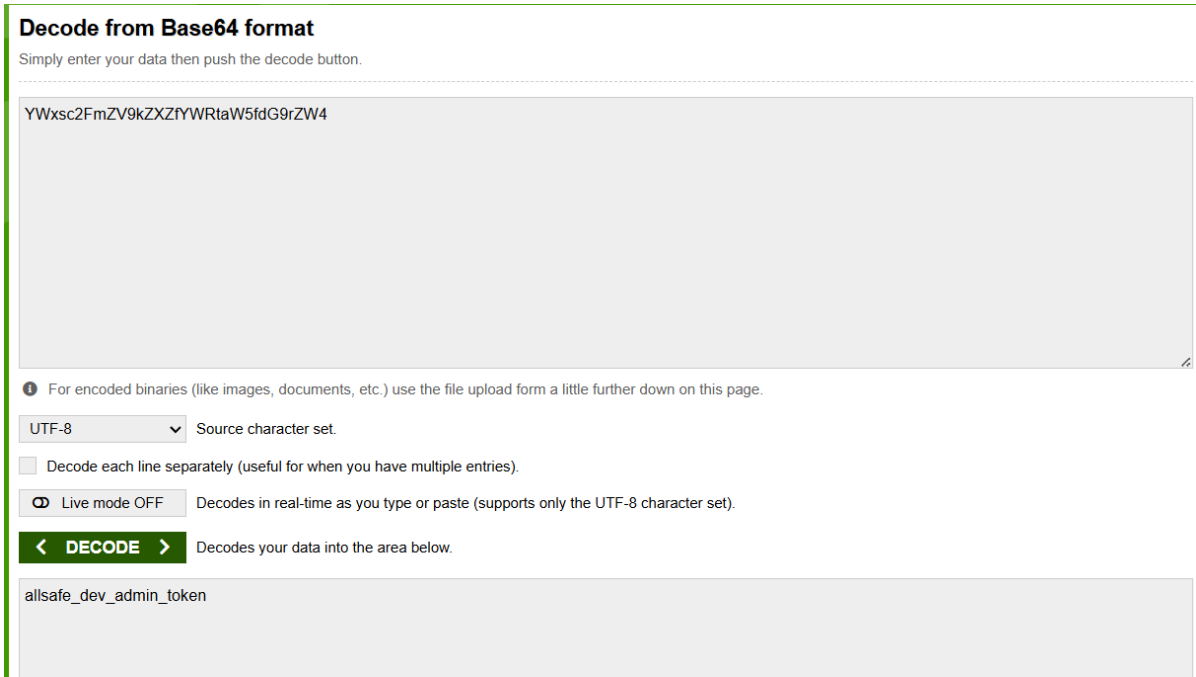
```
10-23 17:10:50.279 6218 6218 D ALLSAFE : http://prod.allsafe.infosecadventures.io/api/v1/note/add?auth_token=YWxsc2FmZV9kZXZfYWRTaW5fdG9rZW4%3D&note=contrase%C3%B1a_123
```

ahora vamos a decodificar, para ello vamos a <https://www.base64decode.org/>

ahi mete el token en mi caso es

auth\_token=YWxsc2FmZV9kZXZfYWRTaW5fdG9rZW4%3D&note=contrase%C3%B1a\_123

copia solo YWxsc2FmZV9kZXZfYWRTaW5fdG9rZW4 y pega en la lista de arriba y dale a decode y te va a salir algo como allsafe\_dev\_admin\_token



**Decode from Base64 format**

Simply enter your data then push the decode button.

YWxsc2FmZV9kZXZfYWRTaW5fdG9rZW4

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

Source character set.

☐ Decode each line separately (useful for when you have multiple entries).

☒ Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

**< DECODE >** Decodes your data into the area below.

allsafe\_dev\_admin\_token

y si en caso si tu texto pegado en aplicacion tiene letras como ñ puedes decodificarlo para ello entra en <https://www.url-encode-decode.com/> y mete tu texto que mi caso es note=contrase%C3%B1a\_123 de cual cogo solo contrase%C3%B1a\_123 y pegamos en la list ade la izquierda y dele click a decode url



y ahí tienes

### Evidencias

#### **Evidencia 1** Comunicación HTTP insegura

impacto: datos transmitidos sin cifrado

Solución: Migrar a HTTPS obligatoriamente, implementar certificate pinning

#### **Evidencia 2** Token de autenticación expuesto

impacto: cualquiera puede suplantar al administrador

Solución: Mover token a headers authorization, usar tokens JWT con expiracion

#### **Evidencia 3** Datos sensibles en texto plano

impacto: Contraseñas y tokens visibles directamente

Solución: Cifrar con AES-256, usar android Keystore para claves

### Severidad

### **Alta.**

El componente Broadcast Receiver de la aplicación procesa y expone información sensible a través de Intents sin control de origen ni autenticación. Esto permite que cualquier aplicación o actor local malicioso envíe o intercepte mensajes internos, accediendo a datos como tokens, contraseñas o URLs de API. Combinado con el uso de HTTP sin cifrado y la exposición del token en texto plano, la vulnerabilidad compromete la confidencialidad y autenticidad de la comunicación entre componentes, pudiendo conducir a suplantación de identidad o robo de información.

### **Mitigación**

- **Restringir el acceso al Broadcast Receiver:**
  - En el `AndroidManifest.xml`, establecer `android:exported="false"` para receptores que no deban recibir Intents externos.
  - Si se requiere comunicación externa, usar permisos personalizados (`android:permission`) para validar el origen.
- **Validar y filtrar los Intents recibidos:** comprobar que los datos provienen de componentes legítimos antes de procesarlos.
- **Eliminar el envío de datos sensibles por Intents o logs:** usar almacenamiento seguro (Android Keystore, EncryptedSharedPreferences).
- **Migrar la comunicación a HTTPS y aplicar Certificate Pinning,** asegurando cifrado de extremo a extremo.

### **Conclusión**

Durante la prueba del componente *Insecure Broadcast Receiver* se observó que la aplicación enviaba información sensible mediante un Intent interceptable, incluyendo un `auth_token` codificado en Base64 y datos transmitidos por HTTP.

Al decodificar el token (YWxsc2FmZV9kZXZfYWRTaW5fdG9rZW4=), se reveló el identificador **allsafe\_dev\_admin\_token**, demostrando exposición total de credenciales administrativas.

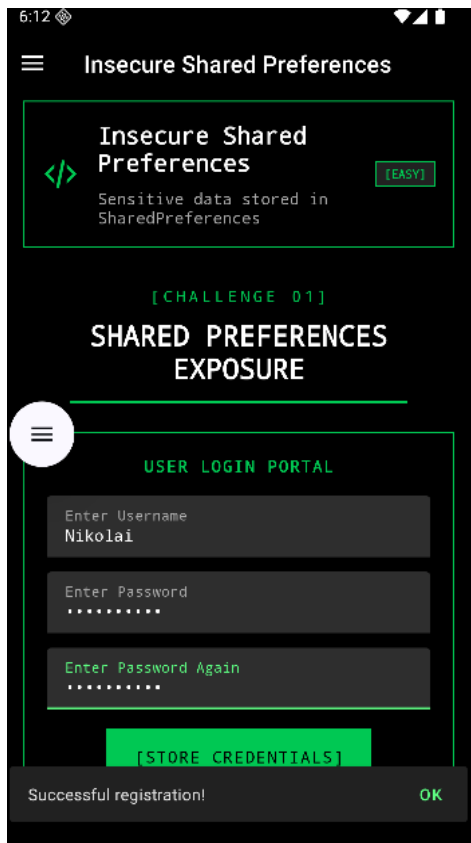
Este comportamiento constituye una **fuga crítica de información** que permitiría a un atacante interceptar, modificar o falsificar peticiones. Se recomienda implementar las medidas de mitigación indicadas y realizar una nueva verificación tras corregir el flujo de comunicación interna.

## Insecure Shared Preferences

### Paso 1:

1. Abrir allsafe
2. Ir a “Insecure Shared Preferences”
3. Crear un usuario y contraseña
  - Usuario Nikolai
  - Contraseña Nikolai123
  - Confirmar contraseña Nikolai123
4. Presionar “STORE CREDENTIALS”





Ahora vamos a la terminal y entramos en nuestro emulador

```
C:\Users\2153760H\Documents\platform-tools-latest-windows\platform-tools>adb shell
```

y ya estando dentro ejecutamos run-as infosecadventures.allsafe

```
emu64xa:/ $ run-as infosecadventures.allsafe
emu64xa:/data/user/0/infosecadventures.allsafe $ |
```

Navegamos a shared preferences

```
cd /data/data/infosecadventures.allsafe/shared_prefs/
```

```
ls -la
```

```
emu64xa:/data/user/0/infosecadventures.allsafe $ cd /data/data/infosecadventures.allsafe/shared_prefs/
emu64xa:/data/data/infosecadventures.allsafe/shared_prefs $ ls -la
total 32
drwxrwx--x  2 u0_a224 u0_a224 4096 2025-10-23 18:12 .
drwx-----  7 u0_a224 u0_a224 4096 2025-10-21 18:59 ..
-rw-rw----  1 u0_a224 u0_a224  953 2025-10-23 13:14 FirebaseHeartBeatW0RFRkFVTFRd+MT05ODM2MzIxNjA2Mjk6YW5kcm9pZDpkMWQ5MTM
yZGRkOTg4ZTcxMjc1NTNj.xml
-rw-rw----  1 u0_a224 u0_a224  163 2025-10-23 18:12 user.xml
emu64xa:/data/data/infosecadventures.allsafe/shared_prefs $ |
```

Veamos el contenido del archivo .xml

cat \*.xml

```
emu64xa:/data/data/infosecadventures.allsafe/shared_prefs $ cat *.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <long name="fire-count" value="3" />
  <string name="last-used-date">2025-10-23</string>
  <set name="device-name/sdk_gphone64_x86_64 android-installer/ android-min-sdk/23 kotlin/2.1.0 fire-rtdb/21.0.0 fire-
app-check/18.0.0 fire-gcs/21.0.1 fire-stg-ktx/21.0.1 fire-android/36 fire-core/21.0.0 fire-db-ktx/21.0.0 device-brand/go
ogle fire-core-ktx/21.0.0 android-platform/ android-target-sdk/35 device-model/emu64xa">
    <string>2025-10-21</string>
    <string>2025-10-23</string>
  </set>
  <set name="device-name/sdk_gphone64_x86_64 android-installer/ android-min-sdk/23 kotlin/2.1.0 fire-rtdb/21.0.0 fire-
app-check/18.0.0 fire-stg-ktx/21.0.1 fire-gcs/21.0.1 fire-android/36 fire-core/21.0.0 fire-db-ktx/21.0.0 device-brand/go
ogle fire-core-ktx/21.0.0 android-platform/ android-target-sdk/35 device-model/emu64xa">
    <string>2025-10-22</string>
  </set>
</map>
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="password">Nikolai123</string>
  <string name="username">Nikolai</string>
</map>
emu64xa:/data/data/infosecadventures.allsafe/shared_prefs $ |
```

Como puedes ver los credencial están en texto plano

```
<map>
  <string name="password">Nikolai123</string>
  <string name="username">Nikolai</string>
</map>
```

### Evidencia 1 Credencia en texto plano

Impacto: Las contraseñas de los usuarios se almacenan sin cifrado, permitiendo que cualquier persona con acceso al dispositivo pueda robarlas fácilmente.

Solución: Implementar EncryptedSharedPreferences, que cifra automáticamente todos los datos antes de guardarlos, para datos muy sensibles como contraseña, es mejor no almacenar localmente y usar tokens de sesión temporales en su lugar.

**Evidencia 2** Archivos accesibles sin root

Impacto: Los archivos de SharedPreferences son legibles sin necesidad de permisos root, exponiendo los datos sensibles a aplicaciones maliciosas básicas.

Solución: Utilizar el modo de almacenamiento privado de la aplicación y evitar complementar los modos world-readable y world-writable, combinar esto con cifrado para una pretensión múltiple.

**Evidencia 3** Falta de cifrado en reposo

Impacto: Los datos permanecen vulnerables incluso cuando la aplicación no se está ejecutando, exponiendo información sensible en dispositivos perdidos o robados.

Solución: Implementar cifrado a nivel de archivo usando android keystore par gestionar las claves de forma segura, esto protege los datos incluso cuando el dispositivo está apagado.

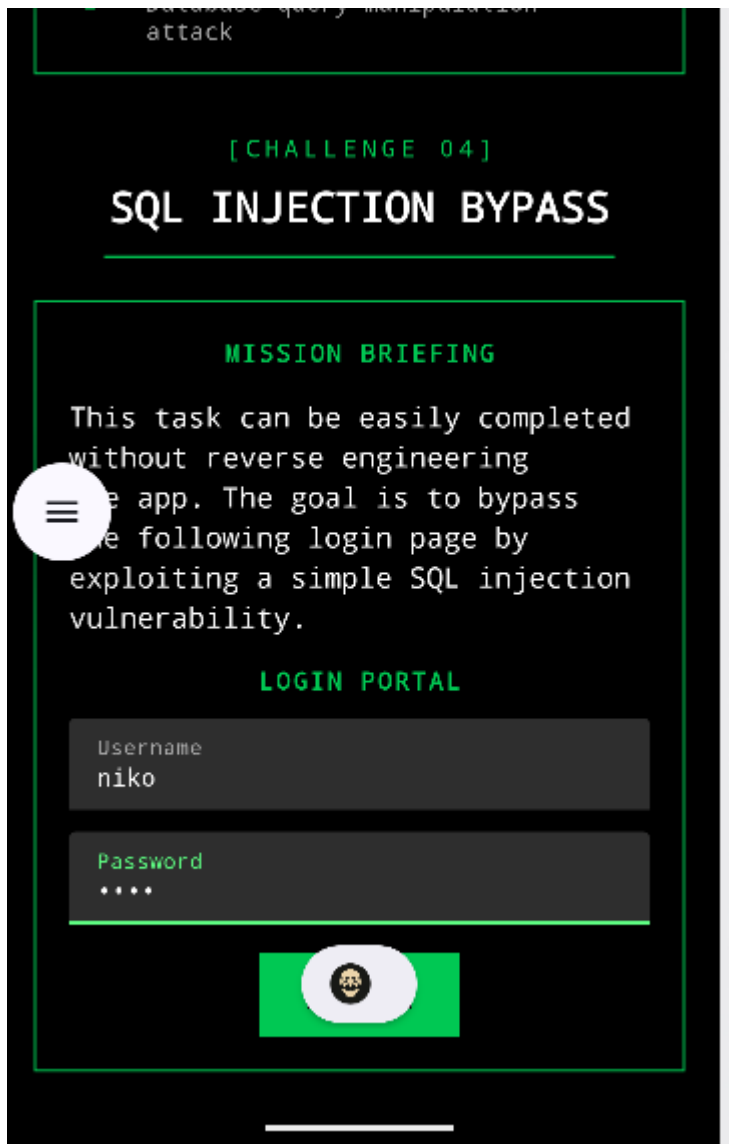
**Evidencia 4** Almacenamiento innecesario de datos sensibles

Impacto: La aplicación almacena contraseñas que podrían evitarse guardar, creando un riesgo de seguridad innecesaria.

Solución: Cambiar el diseño para no almacenar contraseñas locales, en su lugar usar autenticación con tokens que expiran después de un tiempo determinado y pueden revocarse del servidor.

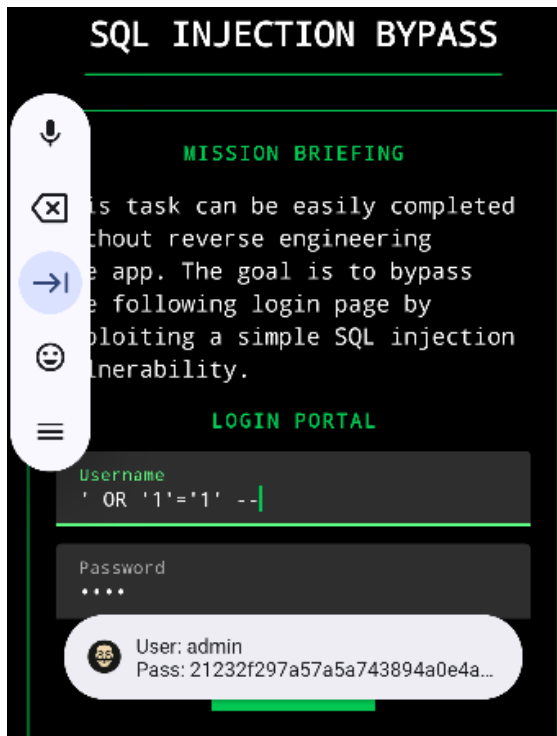
## SQL Injection

Si poner en el campo de usuario y contraseña los caracteres normales no sale nada



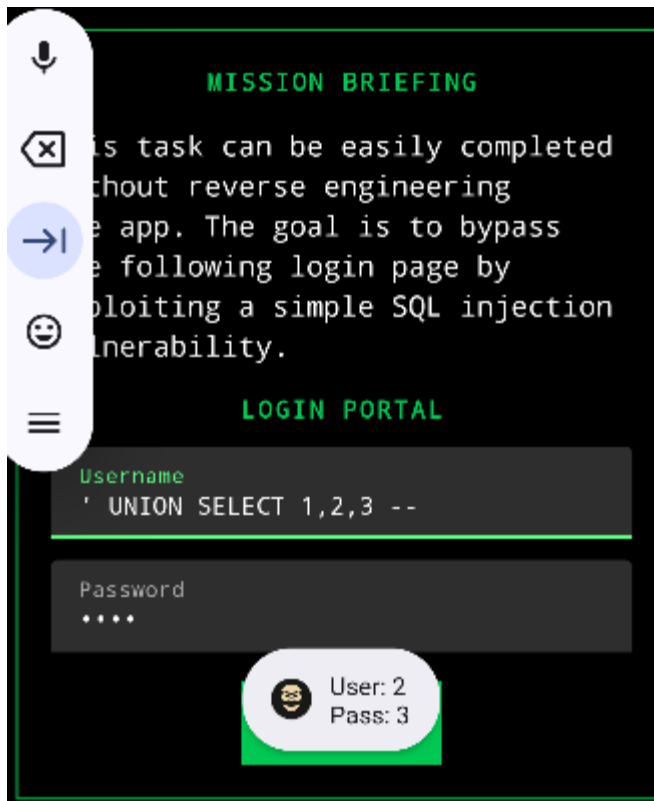
pero si metes en el campo de username por ejemplo

' OR '1'='1' -- , admin' -- o ' OR 1=1 --



ya aparece algo, con que la contraseña es incorrecta

y si ponemos ' UNION SELECT 1,2,3 –



### Explicación de la vulnerabilidad

Causa: La aplicación construye consultas SQL concatenando directamente los inputs del usuario.

Impacto: Permite bypass de autenticación, acceso a datos sensibles, etc.

### Recomendación de mitigación

- Usar consultas parametrizadas o PreparedStatement.
- Validar y sanitizar inputs.
- Usar ORM con parámetros binding.

## Severidad

### Crítica / Alta.

La construcción de consultas SQL mediante concatenación de entradas del usuario permite **inyección SQL** que puede: burlar la autenticación ( ' OR '1'='1' ), extraer datos sensibles mediante **UNION SELECT**, modificar o borrar datos, y en escenarios extremos ejecutar comandos en la BD. Esta vulnerabilidad soporta ataques de escalado muy rápidos y compromete la confidencialidad e integridad de toda la base de datos.

## Mitigación

1. **Usar consultas parametrizadas / PreparedStatement** en todas las llamadas a la BD — nunca concatenar variables de usuario en la cadena SQL.

// ejemplo Java JDBC

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM  
users WHERE username = ? AND password = ?");
```

```
ps.setString(1, username);
```

```
ps.setString(2, password);
```

2. **Validación y saneamiento en servidor:** whitelist de formatos (longitud, caracteres permitidos), rechazar inputs excesivamente largos o con patrones sospechosos; pero **no** confiar sólo en validación cliente — valida siempre en servidor.
3. **Principio de menor privilegio:** la cuenta DB usada por la app debe tener solo los permisos estrictamente necesarios (no DROP/ALTER si no necesario).

4. **Registro y monitorización:** auditar consultas sospechosas, activar alertas para patrones `UNION, --, ' OR 1=1`, múltiples errores de SQL o accesos anómalos.
5. **Pruebas de seguridad continuas:** integrar pruebas de inyección SQL (fuzzing/DAST) en el pipeline de CI/CD.

## Conclusión

Se demostró que la aplicación es vulnerable a inyección SQL (ej.: `' OR '1'='1' --, UNION SELECT`), lo que permite eludir la autenticación y potencialmente extraer o manipular datos críticos. Es imprescindible corregir el código que construye consultas (migrar a PreparedStatements/ORM), endurecer validaciones en el servidor y aplicar controles operativos (menor privilegio, monitorización y pruebas automáticas). La corrección debe considerarse de **alta prioridad** y verificarse mediante pruebas dinámicas posteriores.



#### 4. Tabla Comparativa de Vulnerabilidades

Vulnerabilidad	Severidad	Tipo de Riesgo	Impacto Principal	Mitigación Clave
Hardcoded Credentials	Alta	Exposición de credenciales	Acceso no autorizado	Eliminar credenciales del código
Insecure Logging	Alta	Fuga de información	Exposición de contraseñas	Desactivar logs sensibles

#### 5. Conclusiones Finales

El análisis de la aplicación AllSafe permitió identificar múltiples vulnerabilidades críticas que afectan la confidencialidad, integridad y disponibilidad de los datos.

Las fallas más relevantes son el almacenamiento inseguro de credenciales, el uso de logs con datos sensibles, la ausencia de cifrado en las comunicaciones y la inyección SQL.

Se recomienda aplicar las medidas de mitigación propuestas y realizar una nueva auditoría tras su implementación, para garantizar el cumplimiento de las buenas prácticas definidas en el OWASP MASVS y MSTG.