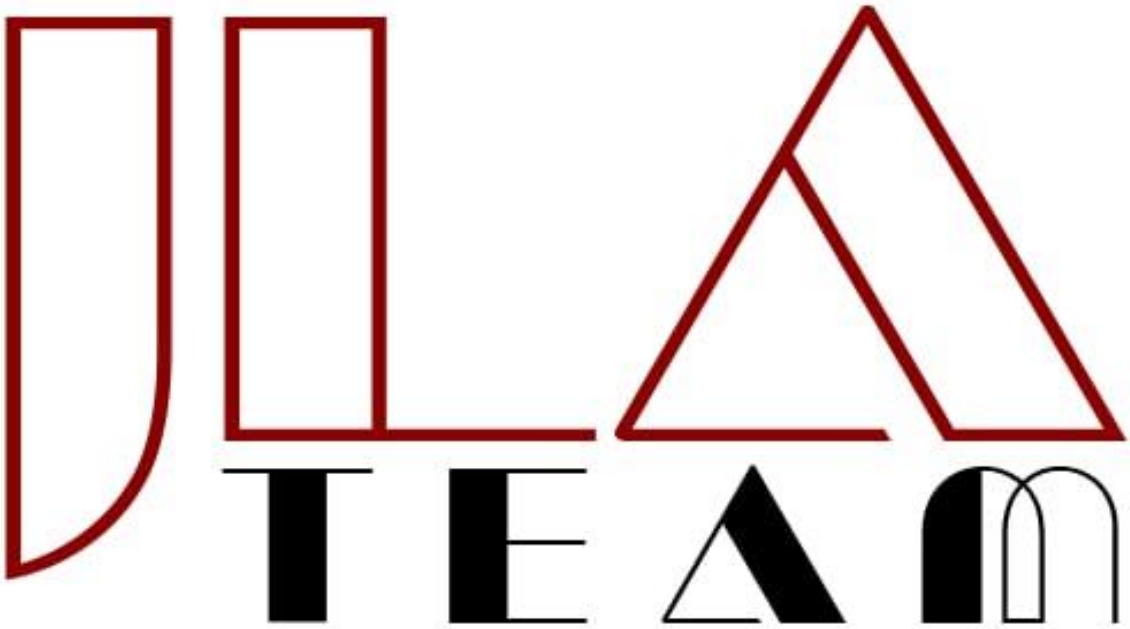



# PENTESTING:

# BITE BURGER

# Metodología

La metodología se basa en enfoques generalmente aceptados en toda la industria para realizar pruebas de penetración para aplicaciones móviles.

Se han buscado, mínimamente, las siguientes vulenrabilidades:

- Falta de protecciones binarias.
- Almacenamiento inseguro de datos.
- Fuga de datos.
- Inyección desde el lado del cliente.
- Encriptación débil.
- Confianza implícita de todos los certificados.
- Ejecución de actividades usando root.
- Exposición de la clave privada
- Exposición de parámetros de la base de datos y peticiones SQL.
- Generadores de números aleatorios inseguros.

# Información

**Aplicación:** Bite Burger **Plataforma:** Android

**Dominio:** 7uzmiokv.directus.app (Alemania) **IP:** 143.204.89.32

**Certificado SHA256:**
d580b43b92eb26b982094284daa8d144f07cd941f857e7896695d056a9df2308

**Vulnerabilidades:**

**Altas:** 6 **Medias:** 7 **Bajas:** 5

# Vulnerabilidades Encontradas

## Alta: Flag de debugging

Habilitado **android:bebuggable= true**, esto permite a un atacante ganar acceso al proceso Java por medio de JWDP y ejecutar código arbitrario. También puede resultar en fugas de información.

Encontrado en el archivo **AndroidManifest.xml**.

```
ICE"/>
ntFactory" android:debuggable="true" android:icon="(
oLQHB8WjyBzZUHq3g"/>
eractive.BiteBurger.MainActivity" android:theme="@s
```

**Recomendación:** Establecer el valor en “false”.

## Alta: Uso de criptografía insegura

Se ha encontrado el uso de criptografía vulnerable, **MD5** y **SHA-1**.

**MD5**

Encontrado en el archivo **com/google/android/gms/measurement/internal/zzky.java**

```
static MessageDigest zzE() {
   for(int var0 = 0; var0 < 2; ++var0) {
      MessageDigest var1;
      try {
         var1 = MessageDigest.getInstance("MD5");
      } catch (NoSuchAlgorithmException var2) {
         continue;
      }

      if (var1 != null) {
```

**SHA-1**

Encontrado en el archivo **com/Google/firebase/messaging/GmsRpc:238**

```
  private String getHashedFirebaseAppName() {
     String var1 = this.app.getName();

     try {
        var1 = base64UrlSafe(MessageDigest.getInstance("SHA-1").digest(var1.getBytes()));
        return var1;
     } catch (NoSuchAlgorithmException var2) {
        return "[HASH-ERROR]";
     }
```

Encontrado en el archivo **com/Google/firebase/installations/local/IidStore:184**

```
private static String getIdFromPublicKey(PublicKey var0) {
   byte[] var4 = var0.getEncoded();

   try {
      MessageDigest var2 = MessageDigest.getInstance("SHA1");
      var4 = var2.digest(var4);
      byte var1 = var4[0];
      var4[0] = (byte)((var1 & 15) + 112 & 255);
      String var5 = Base64.encodeToString(var4, 0, 8, 11);
      return var5;
```

**Recomendación:** En su lugar, utilizar el algoritmo SHA-256 o mejor.

## Alta: Exportación inapropiada de las Actividades de Android

Se ha encontrado una actividad desprotegida:

**androidx.compose.ui.tooling.PreviewActivity (android:exported=true)**

La actividad es compartida en el dispositivo antes de dejarla accesibles a otras aplicaciones.

Por medio de una aplicación maliciosa, un atacante podría insertar otra actividad que permita saltarse autenticaciones o el acceso a zonas restringidas.

## Alta: Exportación inapropiada de Broadcast Receiver

Se ha encontrado que estos procesos están protegidos por un permiso el cual no esta definido. Esto puede provocar que una aplicación maliciosa adquiera el permiso e interactuar con el componente.

Encontrado en:

**com.google.firebase.iid.FirebaseInstanceIdReceiver** con el permiso:

**com.google.android.c2dm.permission.SEND [android:exported=true]**

**androidx.profileinstaller.ProfileInstallReceiver** con el permiso:

**android.permission.DUMP [android:exported=true]**

## Alta: Google API key expuesta en el código

Un atacante podría adquirir las claves del código hacer uso de la API.

Encontrado en el archivo **AndroidManifest.xml**.

```
14    google.android.geo.API_KEY" android:value="AIzaSyBc7ZjCM7NQ1b8pKooLQHB8WjyBzZUHq3g"/>
```

Encontrado en el archivo **res/values/strings.xml**.

```
74        <string name="google_api_key">AIzaSyBO8kWoEtv6wLHrt2MdLEEE2w9nJ8LsIro</string>
```

**Recomendación:** No se debe establecer en el código ninguna clave o credenciales, para evitar que un atacante pueda adquirirlos.

## Alta: Ausencia del archivo de seguridad de la red

No se ha encontrado un archivo de configuración para la red. Al no disponer de tal archivo la aplicación no dispone, por ejemplo, protección contra comunicación con cleartext o la gestión de certificados de confianza.

Este archivo se suele encontrar en la dirección **res/xml/network_security_config.xml**

## Media: Permite realizar Backup

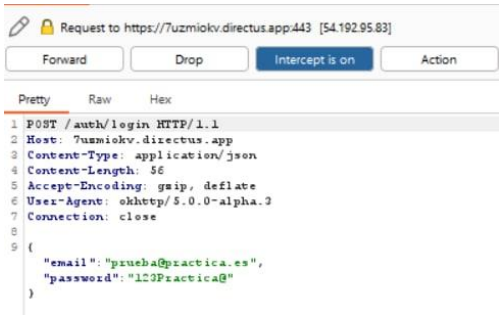
Habilitado **android:allowBackup= true**, permite a un atacante realizar un backup de la aplicación y todos sus datos por medio de herramientas como **adb,** sin necesidad de tener un dispositivo root.

```
<application android:allowBackup="true"
```

**Recomendación:** Establecer el valor en “false”.

## Media: La aplicación permite comunicación con cleartext

No está establecida la opción **android:usesCleartextTraffic="false"** en el archivo **AndroidManifest.xml**. Esto da lugar a que un atacante puede robar información si es capaz de comprobar el tráfico, por ejemplo por medio de un ataque MITM.




**Recomendación:** Establecer en el archivo **AndroidManifest.xml** el parámetro **android:usesCleartextTraffic="false"**.

## Media: Uso de la clase Random en lugar de SecureRandom

El uso de esta clase en relación con la seguridad no se considera seguro, ya que criptográficamente es débil, un atacante podría llegar a averiguarlo.

Encontrado en el archivo **com/Google/Android/gms/measurement/internal/zzky.java**

```
898         long j;
899         if (this.zze.get() == 0) {
900             synchronized (this.zze) {
901                 long nextLong = new java.util.Random(java.lang.System.nanoTime() ^ this.zzs.zzav().currentTimeMillis()).nextLong();
902                 int i = this.zzf + 1;
903                 this.zzf = i;
904                 j = nextLong + i;
```

**Recomendación:** Usar la clase **SecureRandom** en su lugar.

## Media: Protocolo TLS desactualizado

Inicializar SSLContext con una versión genérica o antigua de TLS puede llevar a protocolos de comunicación inseguros.

### okhttp3.internal.platform.BouncyCastlePlatform:38

```
    return var3;
 }

 public SSLContext newSSLContext() {
    SSLContext var1 = SSLContext.getInstance("TLS", this.provider);
    Intrinsics.checkNotNullExpressionValue(var1, "getInstance(\"TLS\", provider)");
    return var1;
 }

 public X509TrustManager platformTrustManager() {
```

### okhttp3.internal.platform.Jdk9Platform:78

```
      Intrinsics.checkNotNullExpressionValue(var3, "getInstance(\"TLS\")");
   } else {
      try {
         var3 = SSLContext.getInstance("TLSv1.3");
      } catch (NoSuchAlgorithmException var2) {
         var3 = SSLContext.getInstance("TLS");
      }
```

### okhttp3.internal.platform.Platform:75

```
    this.log(var3, 5, (Throwable)var2);
 }

 public SSLContext newSSLContext() {
    SSLContext var1 = SSLContext.getInstance("TLS");
    Intrinsics.checkNotNullExpressionValue(var1, "getInstance(\"TLS\")");
    return var1;
 }

 public SSLSocketFactory newSslSocketFactory(X509TrustManager var1) {
```

### okhttp3.internal.platform.ConscryptPlatform:44

```
    return var2;
 }

 public SSLContext newSSLContext() {
    SSLContext var1 = SSLContext.getInstance("TLS", this.provider);
    Intrinsics.checkNotNullExpressionValue(var1, "getInstance(\"TLS\", provider)");
    return var1;
 }

 public SSLSocketFactory newSslSocketFactory(X509TrustManager var1) {
```

### okhttp3.internal.platform.Jdk9Platform:70

```
public SSLContext newSSLContext() {
   Integer var1 = majorVersion;
   SSLContext var3;
   if (var1 != null && var1 >= 9) {
      var3 = SSLContext.getInstance("TLS");
      Intrinsics.checkNotNullExpressionValue(var3, "getInstance(\"TLS\")");
   } else {
      try {
         var3 = SSLContext.getInstance("TLSv1.3");
      } catch (NoSuchAlgorithmException var2) {
```

**Recomendación:** Inicializar SSLContext explícitamente con 'TLSv1.2' or 'TLSv1.3'.

## Media: Vulnerable a Task Hijacking

La aplicación no tiene establecido el parámetro **<activity android:taskAffinity="">**. Esto se define en el archivo **AndroidManifest.xml.**

El atacante necesita tener instalada y corriendo una aplicación maliciosa en el dispositivo. Cuando la víctima inicie la aplicación, la aplicación maliciosa se abrirá en su lugar, esta imitara el login de la aplicación, haciendo que la víctima no sepa que está haciendo login en una aplicación maliciosa.

**Recomendación:** Establecer en el campo **<activity>** del archivo **AndroidManifest.xml** el parámetro **taskAffinity=""** o **singleInstance**.

## Media: Algunos archivos pueden contener información sensible

Información guardada en cleartext en algunos archivos.

Lista en el archivo: **/listas/Cleartext_Archives.txt**

## Media: La aplicación crea archivos temporales

Encontrado en **coil/decode/SourceImageSource.java**

## Baja: Acceso completo a internet

Permite a la aplicación crear sockets de red

## Baja: Puede comprobar el estado de todas las conexiones

## Baja: Evita que el dispositivo entre en modo reposo

## Baja: La aplicación guarda logs

Si se guardan logs, un atacante puede adquirir información, mas importante aun si estos guardan información sensitiva. También, este proceso puede ralentizar el rendimiento de la app.

Lista en el archivo: **/listas/LogsArchives.txt**

## Baja: La aplicación contiene HTTPS URLS en plaintext

El incorporar URLs en el código, da información a un atacante sobre librerías o enpoints. Esta información puede ser utilizada por un atacante para crear aplicaciones de terceros no autorizadas o scrips.

Lista en el archivo: **/listas/Harcoded_URLs.txt**