Manual

SDK Android

PersonalCode Spooff-Offline

Plataforma Android

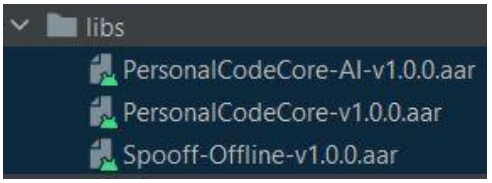
Versión 1.0.0

Manual de usuario para el programador Android de la librería PersonalCodeSpooff-Offline

1 - Importación del SDK al proyecto

1.1 - Copiar los siguientes archivos al proyecto dentro de la carpeta libs:

- PersonalCodeCore-AI-v[VERSIÓN].aar
- PersonalCodeCore-v[VERSIÓN].aar
- Spooff-Offline-v[VERSIÓN].aar



1.2 - Agregar la linea de implementación con su respectiva versión en el archivo build.gradle (Módulo app)

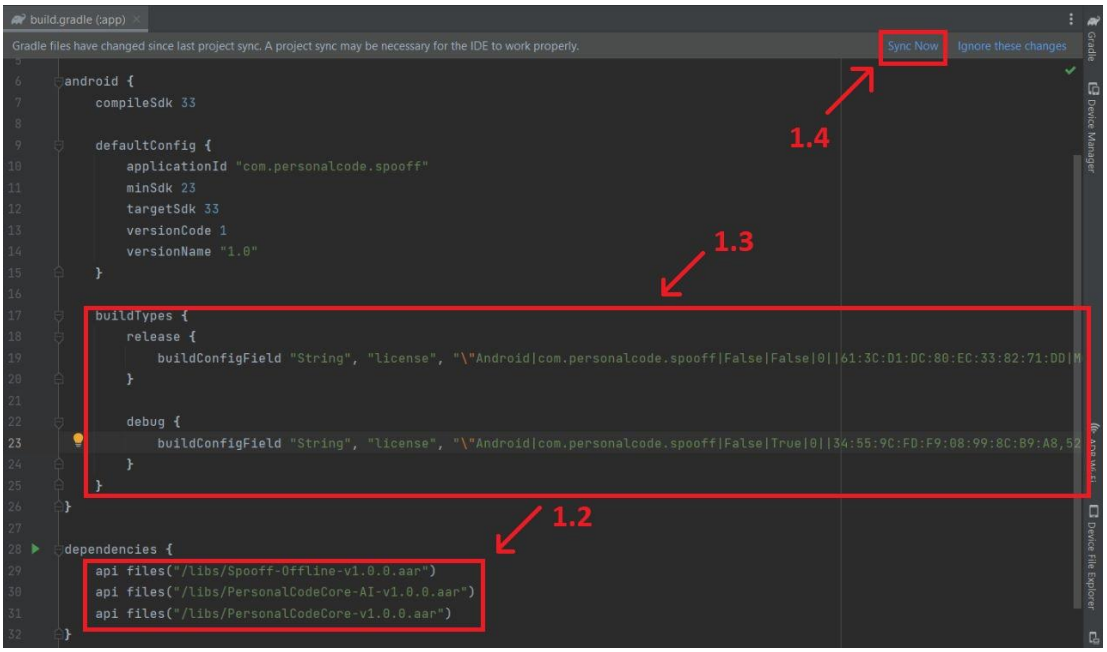
1.3 - Declarar las licencias referentes a Debug y Release en Gradle con las siguientes líneas de código:

```
buildTypes {
    release {
        buildConfigField "String", "license", "\"Android|RELEASE_LICENSE\""
    }
    debug {
        buildConfigField "String", "license", "\"Android|DEBUG_LICENSE\""
    }
}
```

Las licencias que proveemos son firmadas y se compilan junto con la aplicación en su respectivo entorno. Es decir que una vez que se compila en modo Release, la apk no llevará la licencia de Debug y vice-versa. Al ser firmadas las licencias digitalmente, no se podrá cambiar su contenido, por lo que al necesitar algún cambio en la licencia se deberá solicitar dichos cambios directamente al proveedor.

Dichas licencias van vinculadas al applicationId y al SHA1 de compilación de la apk, por lo que es de extrema importancia que se resguarden las llaves de compilación .jks, así como su keystore de Android Studio, de forma que solo los desarrolladores autorizados en las mismas puedan utilizar su licencia.

1.4 - Sincronizar el proyecto



2 - Implementación del SDK al proyecto

2.1 - Importar las siguientes clases a la clase que implementará el SDK:

```
import android.app.Activity
import android.view.ViewGroup
import com.personalcode.spooff.PersonalCodeSpooffOffline
import com.personalcode.spooff.SpooffListener
import com.personalcode.spooff.SpooffResults
```

2.2 - Inicializar una instancia de SpooffListener, donde el SDK notificará sus eventos.

Los métodos *onSpooffCaptured* y *onSpooffCaptureCanceled* cuentan con el tipo de retorno *boolean*, donde se podrá solicitar el cierre de la actividad de Scanner una vez obtenidos los datos mediante el valor true, o simplemente conservarlo abierto con el valor false. En el caso de que se decida conservar abierto, se podrá realizar lo necesario posteriormente mediante el parámetro *activity* recibido en los callbacks. Para el método onSpooffCaptureCanceled cuentan con un status y descripción del posible error que ocasiona que se cierre el SDK, en caso de que el usuario cierre el SDK se mandará un status 100 indicando que no ha sido error alguno.

```
var spooffListener = object: SpooffListener {
    override fun onSpooffCaptured(results: SpooffResults): Boolean {
        Log.d( tag: "SpooffOffline DEMO",  msg: "onSpooffCaptured: ${results.score}")
        return true
    }

    override fun onOpened(activity: Activity?, rootView: ViewGroup?) {
        return
    }

    override fun onSpooffCaptureCanceled(status: Int, description: String): Boolean {
        Log.d( tag: "SpooffOfflineDEMO",  msg: "onSpooffCaptureCanceled: $status $description")
        return true
    }

}
```

2.3 - Inicializar el SDK mediante PersonalCodeSpooffOffline.getInstance(), donde enviamos la instancia de spooffListener que se generó anteriormente y la actividad de la clase.

```
private var spooffOffline: PersonalCodeSpooffOffline? = null
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    if (spooffOffline == null) {
        spooffOffline = PersonalCodeSpooffOffline.getInstance( activity: this, spooffListener)
    }


}
```

2.4 - Validar y solicitar permiso de cámara al usuario.

2.5 - Iniciar el scanner llamando el método *spooffOffline.capture(activity, false)*, donde:

- *Activity* es la instancia de la actividad actual de la aplicación
- El parámetro booleano solicita al SDK que cierre la actividad actual al abrir el scanner o no (true la cierra y false no hará nada)

```
private var spooffOffline: PersonalCodeSpooffOffline? = null
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    btnScanFace = findViewById(R.id.btn_scan_face)

    if (spooffOffline == null) {
        spooffOffline = PersonalCodeSpooffOffline.getInstance( activity: this, spooffListener)
    }

    btnScanFace.setOnClickListener { it: View!
        spooffOffline!!.capture( activity: this,  finishLast: false)
    }

}
```

2.7 - Una vez hecho esto, basta con utilizar el SDK para escanear un rostro para finalmente recibir la respuesta del SDK, en el método *onSpooffCaptured()* de la instancia del interfaz *SpooffListener* declarada en el paso 2.2 de éste documento para que se realice lo necesario con dicha información, la cuál estará almacenada en el objeto p0 del tipo SpooffResults, para acceder al valor liveness devuelto por el servicio basta con acceder a results.score

3.0 - Personalización de la vista del Scanner

Para la personalización del scanner contamos con los siguientes atributos públicos en la clase PersonalCodeSpooffOffline que pueden ser modificados según la necesidad al instanciar la clase (punto 2.3 del presente documento):

```
public Typeface typeface;
public ImageView header;
public ImageView imgLogo;
public ImageButton btnClose;
public ImageButton btnSwitchCam;
```

3.1 - Personalizaciones

Para agregar logo y personalizar más a fondo la vista de *FaceCaptureActivity*, en el callback *onOpened()* de la interfaz *SpooffListener* se obtiene el parámetro *rootView* del tipo *ViewGroup*, donde se podrá implementar nuevas vistas con solo algunas pocas líneas de código.