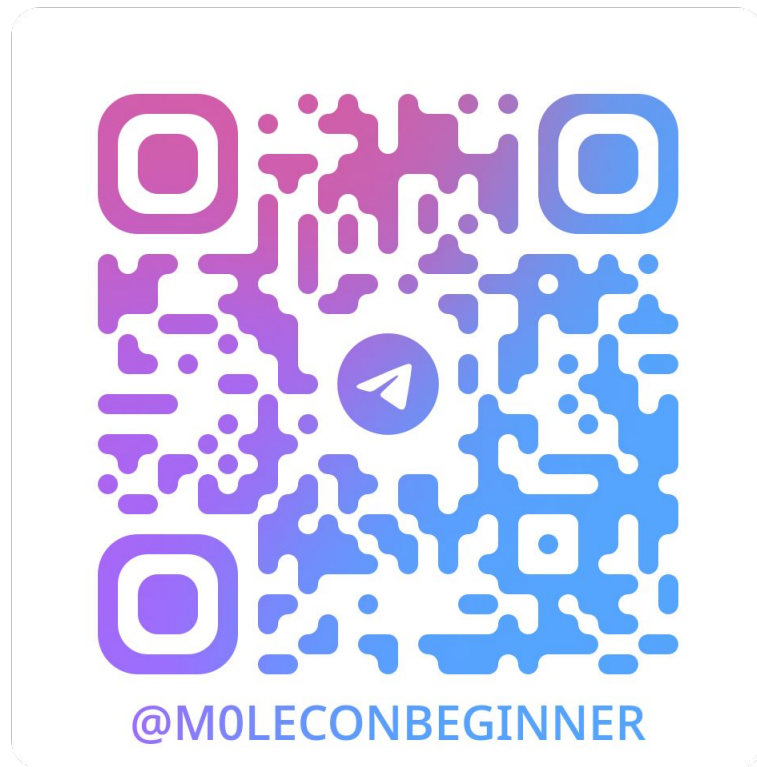


Intro to Android Reverse Engineering

Join the Telegram channel



What does Reverse Engineering mean?

It means **analyzing** for the purpose to **understand** how something works.

This process helps us **finding bugs and vulnerabilities** in a program that we can exploit to find what we need or sometimes even the information we are looking for.

Reverse engineering might allow us to find **useful data** (for example passwords, DLC keys, seeds, etc.) or **unintended behaviours**. In some cases we can even **tamper** some parts of the code.

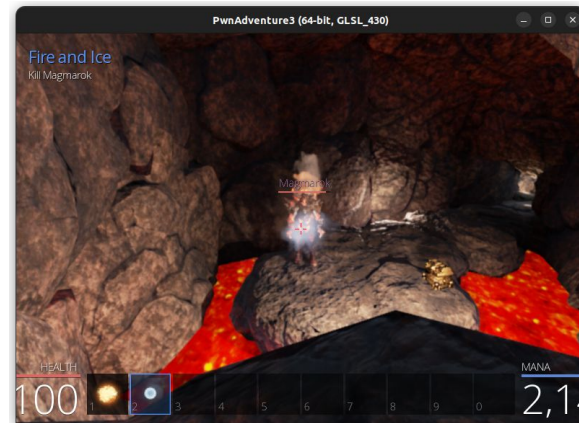
Example:

Buffing our speed in a **videogame** or even implementing fly hacks
(as we have seen in the 2022 workshops!)

A little flashback



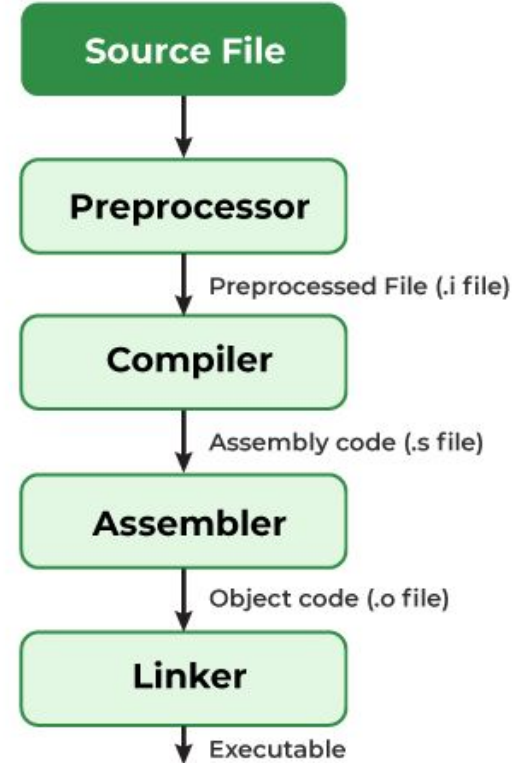
```
mov    rax, 0x3
cvtsi2ss xmm1, rax
mov    rax, 0x2710
cvtsi2ss xmm0, rax
mov    rax, qword [rbp-0x38 {var_40}]
cvtsi2ss xmm2, dword [rax+0x38]
divss  xmm2, xmm0
movss  dword [rbp-0x24 {var_2c_1}], xmm2
cvtsi2ss xmm0, dword [rbp-0x1c {var_24}]
movss  xmm2, dword [rbp-0x24 {var_2c_1}]
movss  dword [rbp-0x3c {var_44_1}], xmm0
movaps  xmm0, xmm2
call   powf
// vita come valore signed
mov    ecx, 10000
mov    rax, 0x4
cvtsi2ss xmm1, rax
movss  xmm2, dword [rbp-0x3c {var_44_1}]
mulss  xmm2, xmm0
mulss  xmm2, xmm1
cvtts2si edx, xmm2
mov    dword [rbp-0x28 {var_30_1}], edx
mov    rax, qword [rbp-0x38 {var_40}]
sub    ecx, dword [rax+0x38]
mov    dword [rbp-0x2c {var_34_1}], ecx
mov    ecx, dword [rbp-0x28 {var_30_1}]
cmp    ecx, dword [rbp-0x2c {var_34_1}]
// jump below or equal = controllo per unsigned int
jbe    0x13afeb
```



From code to executable

In order to be used in **different** machines, a program must be compiled differently according to the architecture and OS.

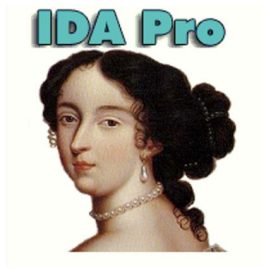
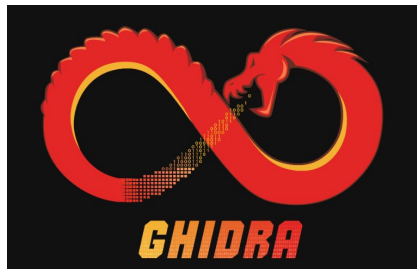
Compilation though is a **lossy** process



Decompilation

Decompiling means trying to obtain a **higher-level** version (the language the original program was written in) of an executable for a better understanding. Since compilation carries a loss of information, decompiling won't always be precise and we'll also rely on understanding the assembly code.

There are various decompilers we might use to analyze our executable, like **Ghidra**, **IDA**, **Binary Ninja** or in case of APKs, **Jadx**.



What is an APK?

Android Package Kit (APK) is the file format for Android apps.

An APK is a type of **archive** format that contain multiple files and metadata in them. They are conceptually a variant of JAR files.

Notable APK concepts:

- Activity → Single focus thing a user can do
- Intent → Object used to request an action from a different component
- Manifest → xml file that contains fundamental metadata for the app execution (list of all activities, required authorizations, etc)



APK

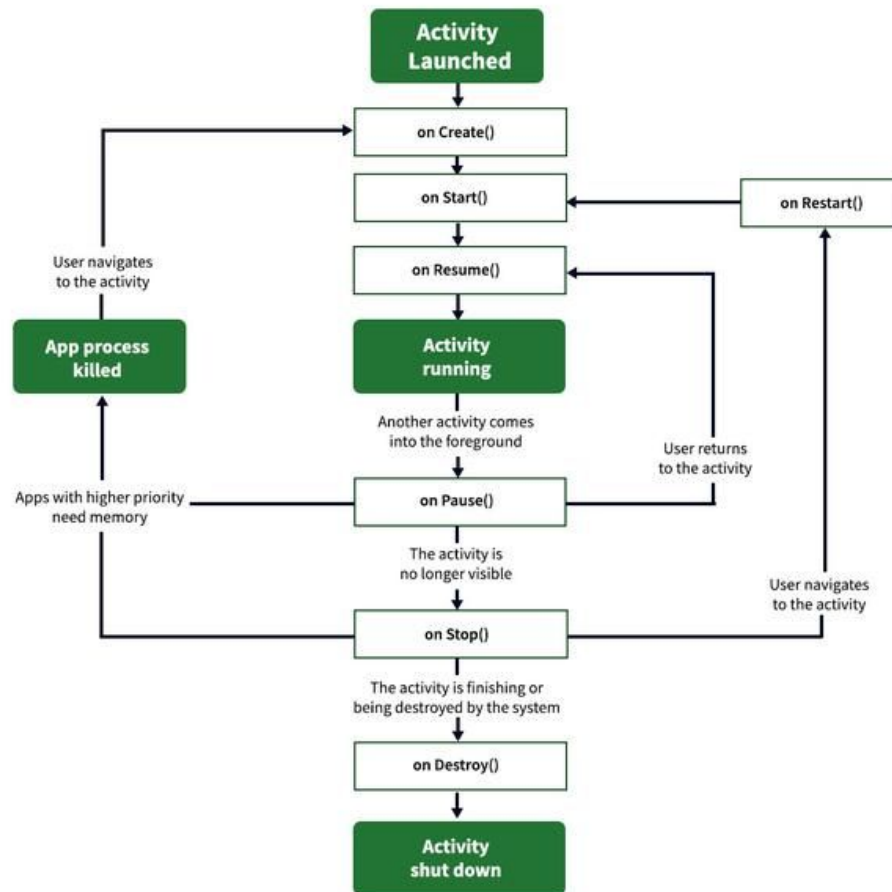
Android manifest

- Presents **essential info** to the android system
- Names the **Java package** for the application
- Describes the components
- Declares both **permissions** necessary to **interact** with other apps and vice versa

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.deneme2.uygulama1">

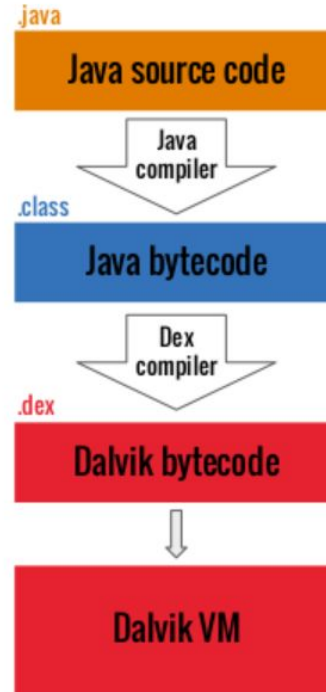
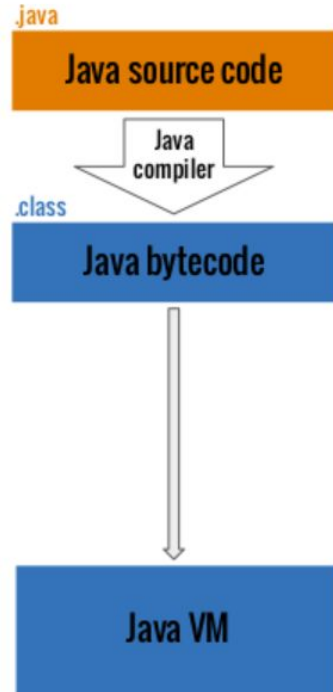
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Uygulama1"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".CustomList"></activity>
    </application>
</manifest>
```

Activity Lifecycle in Android

How an APK is compiled?



NOW SOME CHALLENGES
