1. Unpack the apk file.

```
┌──(kali㉿kali-vm)-[~/Desktop/crackmenative]
└─$ ls
crackme-nat.apk

┌──(kali㉿kali-vm)-[~/Desktop/crackmenative]
└─$ mv crackme-nat.apk crackme-nat.zip

┌──(kali㉿kali-vm)-[~/Desktop/crackmenative]
└─$ unzip crackme-nat.zip
Archive:  crackme-nat.zip
```

2. Convert “classes.dex” to a jar file with dex2jar (https://github.com/pxb1988/dex2jar). *The Error message can be ignored.*

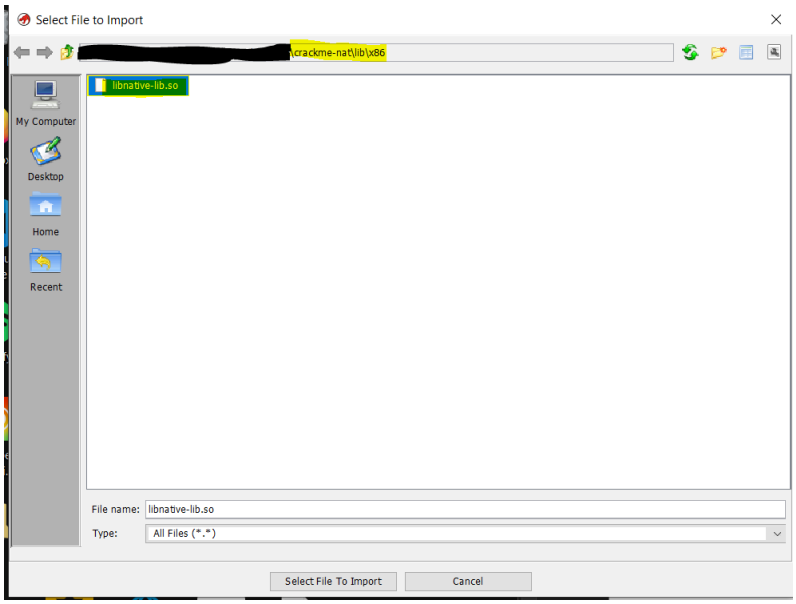
```
┌──(kali㉿kali-vm)-[~/Desktop/crackmenative]
└─$ /opt/dex2jar/d2j-dex2jar.sh classes.dex
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
dex2jar classes.dex → ./classes-dex2jar.jar
Detail Error Information in File ./classes-error.zip
Please report this file to http://code.google.com/p/dex2jar/issues/entry if possible.

┌──(kali㉿kali-vm)-[~/Desktop/crackmenative]
└─$
```

3. Analyze the “LoginViewModel” Class in JD-GUI

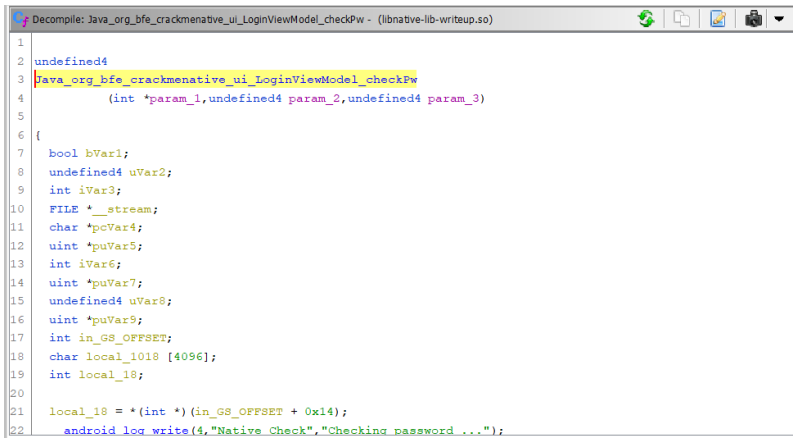
```
private MutableLiveData<LoginResult> loginResult = new MutableLiveData();

static  {
  System.loadLibrary("native-lib");
  x0 = new int[] {
      121, 134, 239, 213, 16, 28, 184, 101, 150, 60,
      170, 49, 159, 189, 241, 146, 141, 22, 205, 223,
      218, 210, 99, 219, 34, 84, 156, 237, 26, 94,
      178, 230, 27, 180, 72, 32, 102, 192, 178, 234,
      228, 38, 37, 142, 242, 142, 133, 159, 142, 33 };
}

private boolean isPasswordValid(String paramString) { return (paramString != null && paramString.tri

public native boolean checkHooking();

public native int[] checkPw(int[] paramArrayOfInt);

protected int[] getCode(String paramString) {
  byte[] arrayOfByte = paramString.getBytes();
  int[] arrayOfInt = new int[paramString.length()];
  for (byte b = 0; b < paramString.length(); b++)
    arrayOfInt[b] = arrayOfByte[b] ^ x0[b];
  return arrayOfInt;
}

LiveData<LoginFormState> getLoginFormState() { return this.loginFormState; }

LiveData<LoginResult> getLoginResult() { return this.loginResult; }

protected String getStringFromCode(int[] paramArrayOfInt) {
  byte[] arrayOfByte = new byte[paramArrayOfInt.length];
  for (byte b = 0; b < paramArrayOfInt.length; b++)
    arrayOfByte[b] = (byte)(paramArrayOfInt[b] ^ x0[b]);
  return new String(arrayOfByte);
}

public void login(String paramString) {
  if (checkHooking()) {
    this.loginResult.setValue(new LoginResult(Integer.valueOf(2131624024)));
    return;
```

We can see that the CheckPw() is a native method due to the *native* keyword. The library is loaded from “native-lib” (file: *libnative-lib.so*).

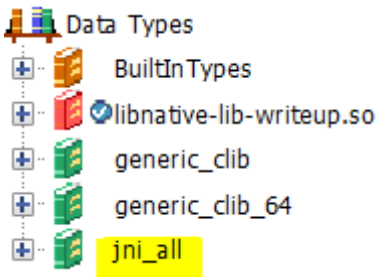
4. Load the library & analyze into Ghidra (https://ghidra-sre.org/)
   I have blacked out the beginning of the path to not expose my username.



5. Open the CheckPw function



6. To change the function signature, add the JNI (Java Native Interface) Data Types to Ghidra (https://github.com/Ayrx/JNIAnalyzer/blob/master/JNIAnalyzer/data/jni_all.gdt)




```
undefined4 Java_org_bfe_crackmenative_ui_LoginViewModel_checkPw (JNIEnv
* JNI, jobject param_2, jintArray intArr)
```

7. The lines 24 & 26 tell us, that the length of the flag must be 27 (hex: 0x1b)

```
21   local_18 = *(int *)(in_GS_OFFSET + 0x14);
22   __android_log_write(4,"Native Check","Checking password ...");
23   p_Var2 = (*(*JNI)->NewIntArray)(JNI,0);
24   jVar3 = (*(*JNI)->GetArrayLength)(JNI,(jarray)intArr);
25   p_Var9 = p_Var2;
26   if (jVar3 == 0x1b) {
27     __stream = fopen("/proc/self/maps","r");
```

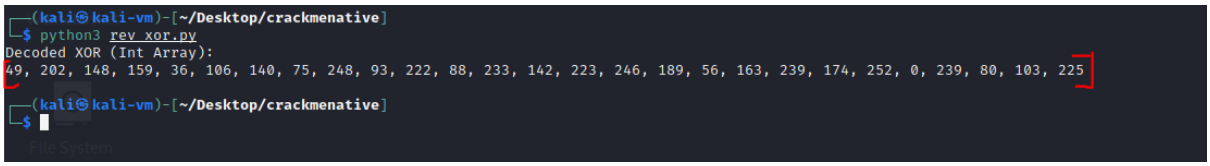
8. The code block (lines 47 – 60) tell us the following:
   a. A pointer to the Array elements is created (pjVar5)
   b. 3 Variables are loaded (puVar6, puVar8, puVar10) from a respective pointer based off the iterator (iVar7 + 108 [hex: 0x6c])
   c. The Loop continues, while: *puVar10 ^ pjVar5 + iterator ^ *puVar8) != puVar6*

```
46     if (!bVar1) {
47       pjVar5 = (*(*JNI)->GetIntArrayElements)(JNI,intArr,(jboolean *)0x0);
48       puVar6 = &DAT_00010b80;
49       puVar8 = &DAT_00010b4c;
50       puVar10 = &DAT_000109e4;
51       iVar7 = -0x6c;
52       do {
53         p_Var9 = p_Var2;
54         if ((*puVar10 ^ *(uint *)((int)pjVar5 + iVar7 + 0x6c) ^ *puVar8) != *puVar6) break;
55         puVar6 = puVar6 + 1;
56         puVar8 = puVar8 + -1;
57         puVar10 = puVar10 + 1;
58         iVar7 = iVar7 + 4;
59         p_Var9 = intArr;
60       } while (iVar7 != 0);
61     }
62   }
```

9. We can take the first 27 hex values (without hex 00) starting from the Addresses 00010b80 (puVar6) & 000109e4 (puVar10). We can do the same in **reverse** for the Address 00010b4c (puVar8).

```
00010b80  80 00 00 00 e3 00 00 00 da 00 00 00 c7 00 00 00
00010b90  2e 00 00 00 f1 00 00 00 a2 00 00 00 91 00 00 00
00010ba0  6b 00 00 00 dc 00 00 00 6b 00 00 00 b5 00 00 00
00010bb0  e5 00 00 00 af 00 00 00 3f 00 00 00 b9 00 00 00
00010bc0  ee 00 00 00 5b 00 00 00 26 00 00 00 92 00 00 00
00010bd0  66 00 00 00 c5 00 00 00 cb 00 00 00 de 00 00 00
00010be0  81 00 00 00 79 00 00 00 da 00 00 00

000109e0  64 61 00 00 d0 00 00 00 45 00 00 00 28 00 00 00
000109f0  76 00 00 00 6f 00 00 00 f3 00 00 00 5a 00 00 00
00010a00  f4 00 00 00 c7 00 00 00 ce 00 00 00 fb 00 00 00
00010a10  c3 00 00 00 7f 00 00 00 48 00 00 00 ce 00 00 00
00010a20  3c 00 00 00 3a 00 00 00 0b 00 00 00 f1 00 00 00
00010a30  53 00 00 00 b1 00 00 00 4b 00 00 00 b9 00 00 00
00010a40  5e 00 00 00 a2 00 00 00 65 00 00 00 77 00 00 00

00010ae0  48 00 00 00 4c 00 00 00 7b 00 00 00 73 00 00 00
00010af0  6f 00 00 00 72 00 00 00 72 00 00 00 79 00 00 00
00010b00  2e 00 00 00 74 00 00 00 68 00 00 00 69 00 00 00
00010b10  73 00 00 00 2e 00 00 00 69 00 00 00 73 00 00 00
00010b20  2e 00 00 00 4e 00 00 00 4f 00 00 00 54 00 00 00
00010b30  2e 00 00 00 74 00 00 00 68 00 00 00 65 00 00 00
00010b40  2e 00 00 00 66 00 00 00 6c 00 00 00 61 00 00 00
```

10. Parse the hex sequence into the python script (rev_xor.py) & get the output.



11. The output of the python script is an integer Array which must be inserted into the Java program (decipher2.java) / the Integer Array (**x0**) to output the decoded flag.
The code was copied from the method in the LoginViewModel.