1. Unpack (unzip) & convert the classes.dex file to a .jar file with **dex2jar** (as done in previous write-ups).

```
┌──(kali㉿kali)-[~/Desktop/crackme-dyn]
└─$ cp crackmedyn.apk crackmedyn.zip
```

```
┌──(kali㉿kali)-[~/Desktop/crackme-dyn]
└─$ unzip crackmedyn.zip
```

```
┌──(kali㉿kali)-[~/Desktop/crackme-dyn/crackmedyn]
└─$ d2j-dex2jar classes.dex
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
dex2jar classes.dex → ./classes-dex2jar.jar
```

2. Open the .jar file in jd-gui. We can inspect the LoginViewModel class.
   We find out the following:
   a. A temporary file is created, and the dynamic code is written to it.
   b. The “DexClassLoader” is initialized. This lets us assume that the temporary file created is an .apk file that contains a classes.dex and its classes are loaded into “DexClassLoader”. (*Source: https://developer.android.com/reference/dalvik/system/DexClassLoader*)
   c. The temporary file is deleted.
   d. The DexClassLoader loads the class “org.bfe.bootstrap.HLLoginCheck” and invokes the “checkPin” Method.

```
private void loadVerifier() {
  try {
    InputStream inputStream = this.context.getResources().openRawResource(2131623936);
    File file = File.createTempFile("code", "tmp", this.context.getCacheDir());
    AESUtil.writeDecryptedFile(inputStream, file);
    DexClassLoader dexClassLoader1 = new DexClassLoader();
    this(file.getPath(), file.getParent(), null, this.context.getClassLoader());
    this.dexClassLoader = dexClassLoader1;
    file.delete();
  } catch (Exception exception) {
    Log.e("Main", "Failed to load Verifyer.");
  }
}

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

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

public void login(String paramString) {
  try {
    Class clazz = this.dexClassLoader.loadClass("org.bfe.bootstrap.HLLoginCheck");
    String str = (String)clazz.getDeclaredMethod("checkPin", new Class[] { String.class }).invoke(clazz, new Object[] {
    if (str != null && str.length() > 1) {
      LoggedInUser loggedInUser = new LoggedInUser();
      this(str, "Well done you did it.");
      MutableLiveData mutableLiveData = this.loginResult;
      LoginResult loginResult1 = new LoginResult();
      this(loggedInUser);
      mutableLiveData.setValue(loginResult1);
    } else {
      MutableLiveData mutableLiveData = this.loginResult;
      LoginResult loginResult1 = new LoginResult();
      this(Integer.valueOf(2131689568));
      mutableLiveData.setValue(loginResult1);
    }
  } catch (Exception paramString) {
    this.loginResult.setValue(new LoginResult(Integer.valueOf(2131689513)));
  }
}
```

3. To prevent the deletion of the file we can patch, repackage & resign the apk file. The process is described in the "Patching, Repackaging, and Re-Signing" section of this README from OWASP: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05c-Reverse-Engineering-and-Tampering.md
   We will patch the loadVerifier method and replace the file.delete() with file.exists(), which will have the same return value.
   Important: When unpacking & disassembling the original .apk file, use the command "apktool d ***–no-res*** *<filename>.apk". Otherwise, the repackaging with "apktool b" will fail.*

```
.line 58
invoke-virtual {v1}, Ljava/io/File;→delete()Z
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
```

```
.line 58
invoke-virtual {v1}, Ljava/io/File;→exists()Z
:try_end_0
.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
```

4. After repackaging and resigning, we can run the apk in Android Studio (Android Virtual Device) & inspect the logs with "adb logcat". A warning message will tell us the location of the temporary file.

```
05-09 17:04:19.659 10162 10162 W art     : Failed to determine odex file name: Dex location /data/user/0/or
g.bfe.crackme/cache/code468958016tmp has no extension.
```

5. Use adb pull to pull the file from the device to our local file system.
   The target directory is a shared folder with the Kali VM.

```
adb pull /data/user/0/org.bfe.crackme/cache/code468958016tmp %USERPROFILE%\Desktop\SHC\crackme-dyn
p: 1 file pulled, 0 skipped. 0.2 MB/s (2285 bytes in 0.012s)
```

6. Perform the procedures in **Step 1** of the writeup with the pulled file.

```
┌──(kali㉿kali)-[~/Desktop/crackme-dyn]
└─$ unzip code468958016tmp
```

```
┌──(kali㉿kali)-[~/Desktop/crackme-dyn]
└─$ d2j-dex2jar classes.dex
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
dex2jar classes.dex → ./classes-dex2jar.jar
```

7. Analyze the HLLoginCheck class. We can see that the modulo of a number (25239776756291) and the input pin is checked against 0.

```
private static String checkPin(int paramInt) throws InterruptedException {
  BigInteger bigInteger = new BigInteger(String.format("%d", new Object[] { Integer.valueOf(paramInt) }));
  if ((new BigInteger("25239776756291")).mod(bigInteger).intValue() != 0) {
    long l = countWrongTries;
    Thread.sleep(l * l * 200L);
    countWrongTries++;
    Log.e("Login", "Wrong PIN.");
    return null;
  }
  countWrongTries = 0L;
  return getFlag(bigInteger.multiply(bigInteger).toString());
}

public static String checkPin(String paramString) {
  if (paramString != null)
    try {
      if (paramString.length() >= 4) {
        Integer integer = Integer.valueOf(Integer.parseInt(paramString));
        StringBuilder stringBuilder = new StringBuilder();
        this();
        stringBuilder.append("Pin is: ");
        stringBuilder.append(integer);
        Log.i("Login", stringBuilder.toString());
        return checkPin(integer.intValue());
      }
    } catch (Exception paramString) {}
  Log.e("Login", "Pin must be an Integer of 6 digits.");
  return null;
}
```

8. From the previous step we know that the modulo of 2 numbers (known number & input pin) is checked against 0. This means we can Bruteforce the possible 6-digit pin and check if the modulo is 0. This has been done in a python script (*solve.py*).
   Run it & get the correct code.

```
Trying pin:  777734
Trying pin:  777735
Trying pin:  777736
Trying pin:  777737
Correct pin: 777737
```

9. Enter the number into the app & get the flag.