WAYNE STATE UNIVERSITY
COLLEGE OF ENGINEERING
Department of Computer: Cyber Security Practice

Lab 5: Android Application Reverse Engineering and Obfuscation

Introduction

Reverse Engineering is a popular hacking approach that extracts the knowledge and design of a system and reproduces its behavior based on the extracted information. To prevent the reverse engineering, we often use the obfuscation to raise the bar of the difficulty of this process.

In this lab, you will learn how to do reverse engineering and obfuscation for Android applications. First, you will need to write a simple Android application. Then, you will reverse the byte code of the application by adding a malicious function. Lastly, you will use packers to obfuscate the application and protect it from reverse engineering.

Software Requirements

All required files are packed and configured in the provided virtual machine image.

- The VMWare Software
  http://apps.eng.wayne.edu/MPStudents/Dreamspark.aspx

- The Kali Linux, **64 bit**, Penetration Testing Distribution
  https://www.kali.org/downloads/

- Java SE Development Kit (JDK)
  http://www.oracle.com/technetwork/java/javase/downloads/index.html

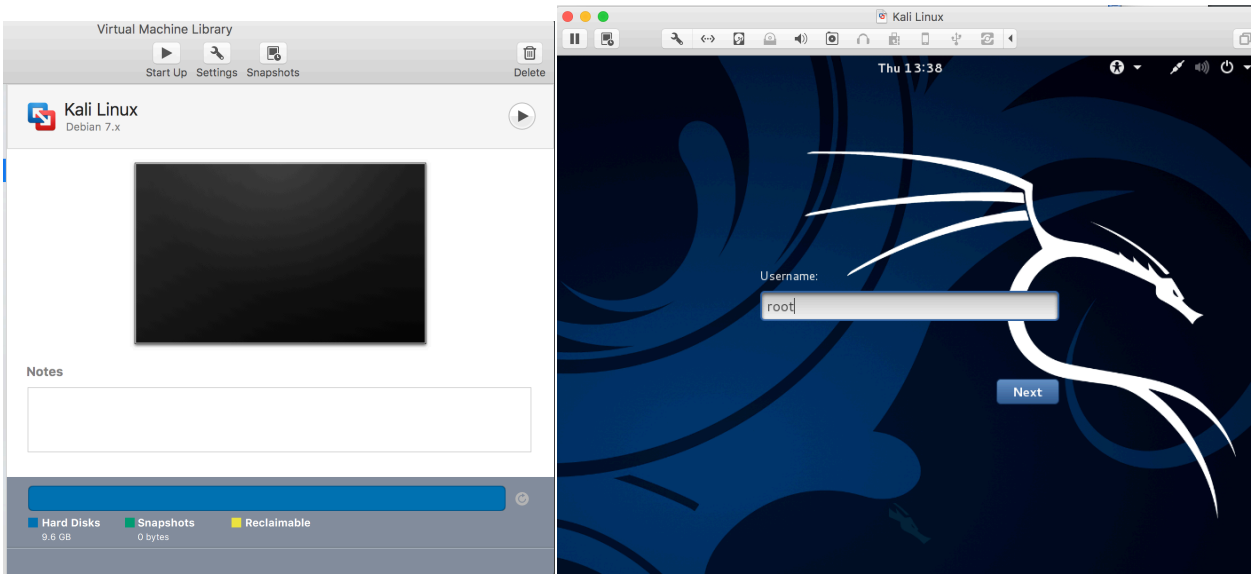
- Android Software Development Kit (SDK)
  http://developer.android.com/sdk/index.html

- Android Studio Integrated Development Environment (IDE) or Eclipse
  http://developer.android.com/sdk/index.html

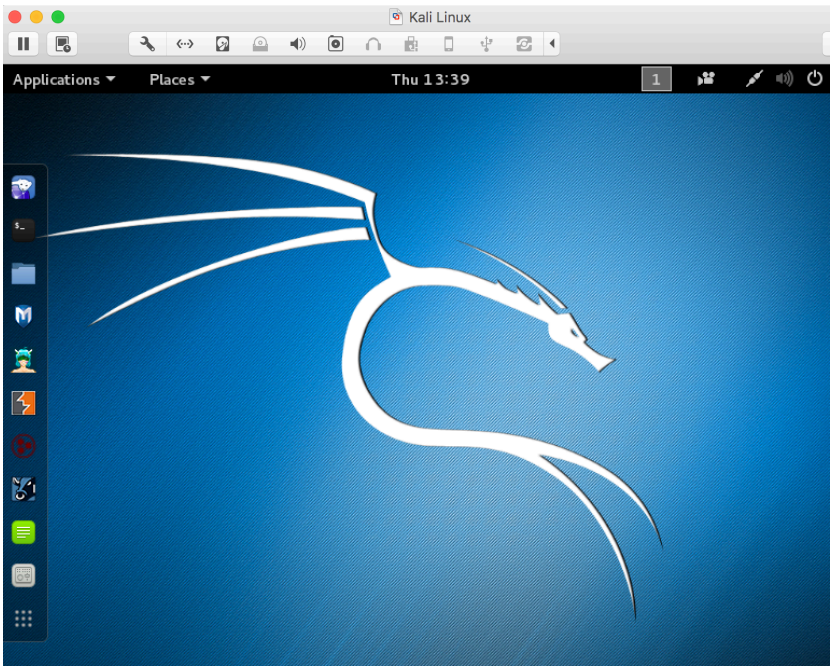
- Android Emulator System or Android Device
  http://developer.android.com/sdk/index.html

Starting the Lab 5 Virtual Machine

The Kai Linux VM has all the required files. Select the VM named “Lab5”. Due to performance limitation, I recommend that you work on the lab assignment on your laptop without virtual machine.



Login the Kali Linux with username root, and password [TBA in the class]. Below is the screen snapshot after login.



Setting up the Environment

This lab requires a few tools to be installed. Fortunately, I have installed and setup these tools for you. If you want to finish this lab on your own laptop or desktops without virtual machine, please follow the steps below to setup the environment.

Install Java SE Development Kit (JDK)

First, you need to install the Java SE Development Kit (JDK) on your system. After you install the JDK, this should also provide the ***jarsigner*** and ***keytool*** utilities. The latest JDK can be found at the following URL:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

The JDK supports multiple OSes including Windows, Linux, and Mac OS. If you use Kali Linux, you need to extract the JDK folder and switch the default OpenJDK to Java SE JDK. The instructions for that can be find here:

http://www.blackmoreops.com/2015/08/15/how-to-install-java-in-kali-linux-2-0-kali-sana/

Install Android Studio

Android Studio includes Android Integrated Development Environment (IDE), Software Development Kit (SDK) tools, and an emulator system. You can download it from the following URL:

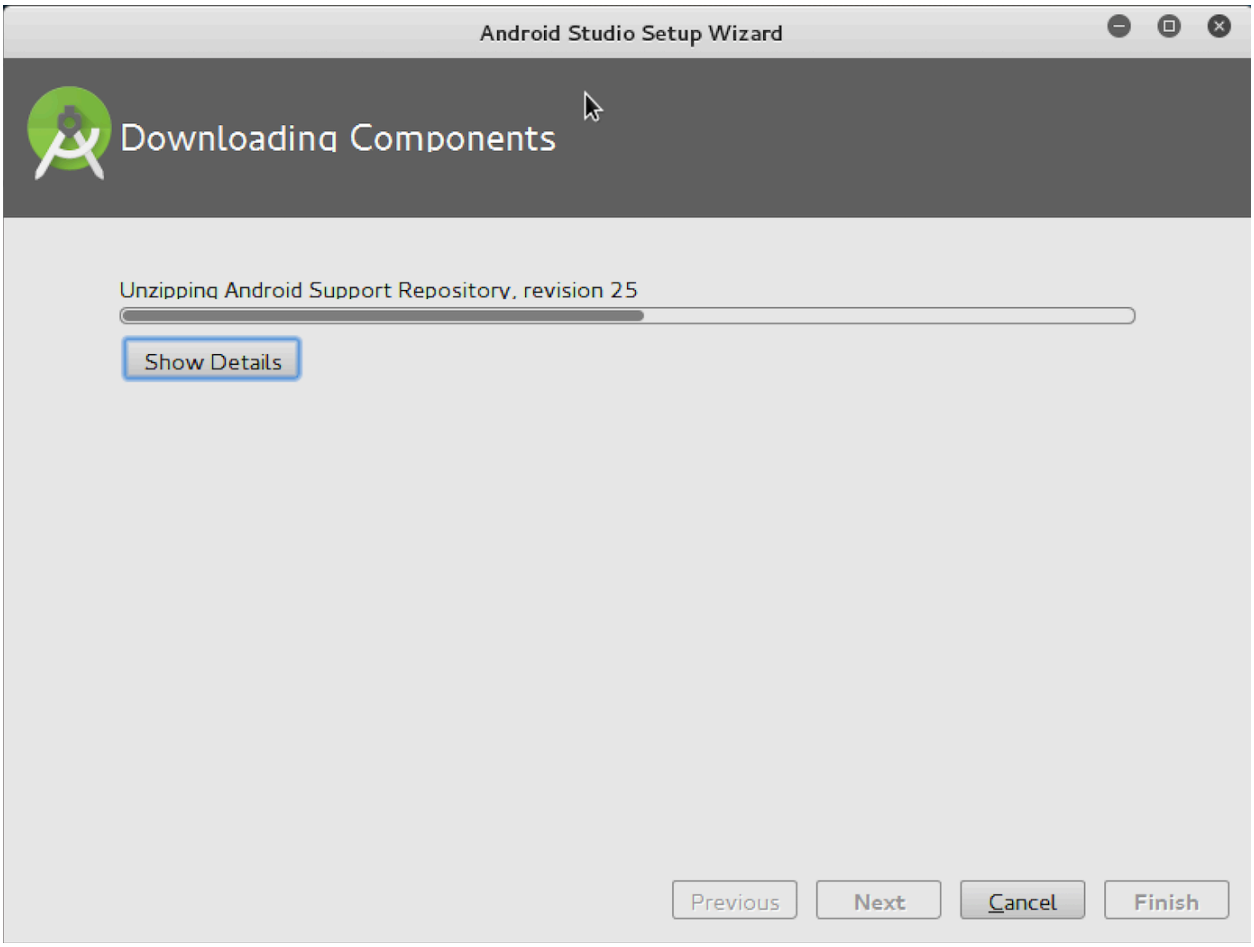
http://developer.android.com/sdk/index.html

Then you extract the android-studio file. In our Kali Linux image, I extracted under */root/Documents/android-studio*. To run the Android Studio, just go to the *bin* directory and run *studio.sh*.

```
$ cd /root/Documents/android-studio/bin

$ ./studio.sh
```

If you run the studio.sh at the first time, it would setup the Android SDK for you. The screenshot below shows the Android Studio Setup Wizard when running at the first time.



Android SDK Tools

In our Kali Linux image, the Android SDK is at */root/Android/Sdk.* All of the SDK tools are under this directory. Below are some tools that you will use in the lab.

$ android sdk*:* Android SDK manager; it is under Android/Sdk/tools/ directory

$ android avd*:* Android Virtual Device manager; it is under Andorid/Sdk/tools/ directory; you can create an virtual device using the AVD manager.

$ emulator -avd <avd_name> [<options>]*:* Start an emulator with a configured virtual device; it is under the Andorid/Sdk/tools/ directory

$ adb*:* Android Debug Bridge. You can gain a shell by running $ adb shell. It is under Android/Sdk/platform-tools/

$ zipalign: Optimizes .apk files by ensuring that all uncompressed data starts with a particular alignment. It is under the Android/Sdk/build-tools/23.0.2/ directory.

You can find a full list of SDK tools from here:

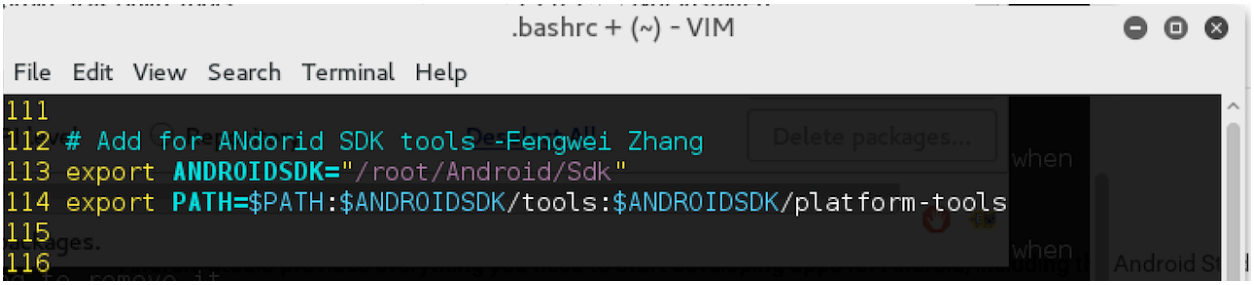
http://developer.android.com/tools/help/index.html

For easy access to the SDK tools from a command line, we add the location of the SDK's tools/, platform-tools/, and build-tools/ to your ***PATH*** environment variable.

$ vim ~/.bashrc

$ [Edit the file as the screenshot below]

$ source ~/.bashrc



Android Studio IDE vs. Eclipse IDE

For this lab, you can use either Android Studio IDE or Eclipse IDE. To use Eclipse, you need to install the Android ADT plugins. In this lab instruction, I will use Android Studio IDE as an example.

Android Emulator vs. Android Device

You can use a real android device to test the applications in this lab. If you do not have an Android device or do not want to use your Android device, then you can use the emulator. The instructions for creating and using the emulator are at the URL below.

http://developer.android.com/tools/devices/emulator.html

Creating an Android Application

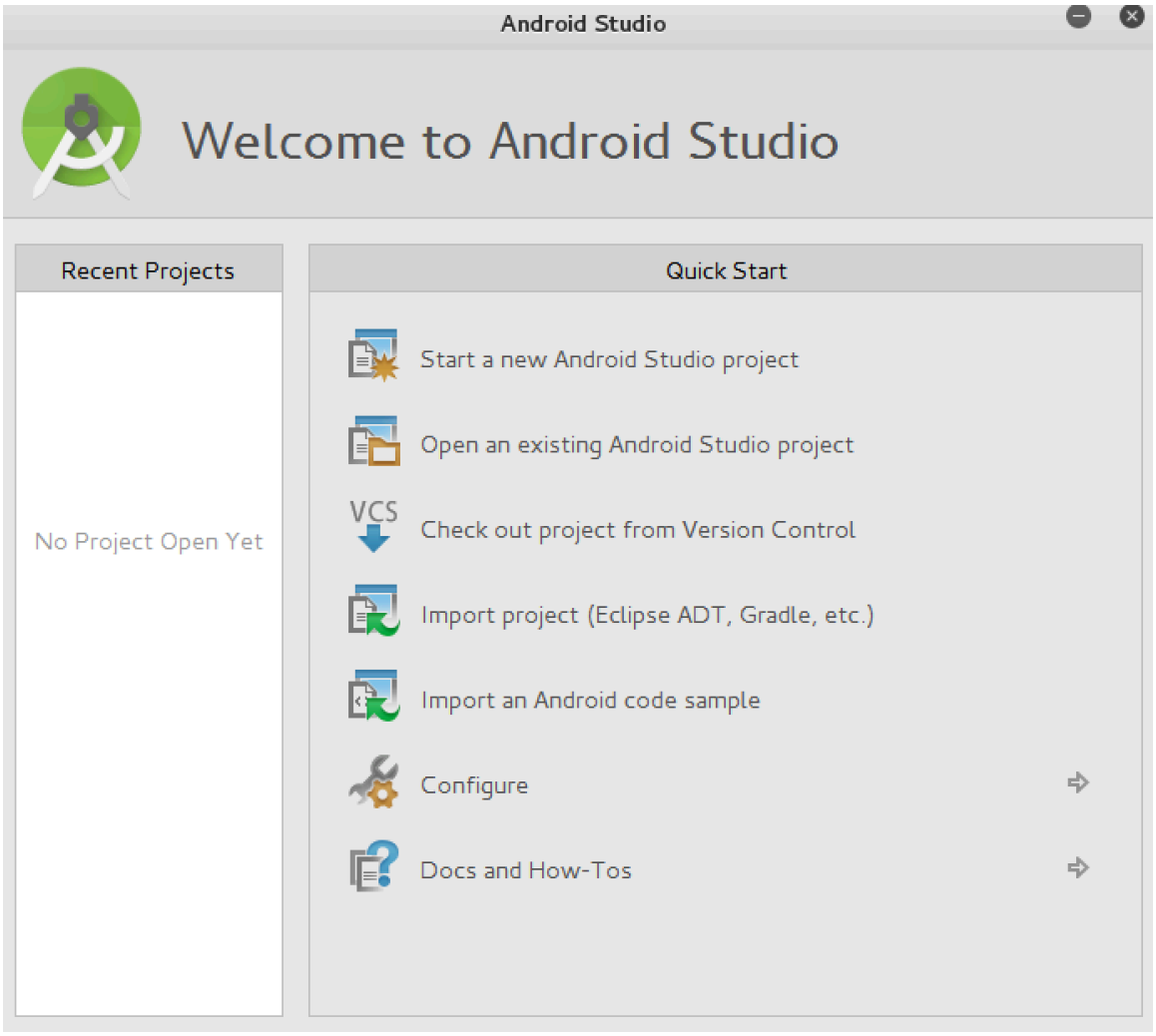
The goal of this exercise is to familiarize you with Android development and get started with a simple Login application. The following are the steps to build this app. You also can find similar instructions from here:

http://developer.android.com/training/basics/firstapp/creating-project.html

1. Start the Android Studio IDE

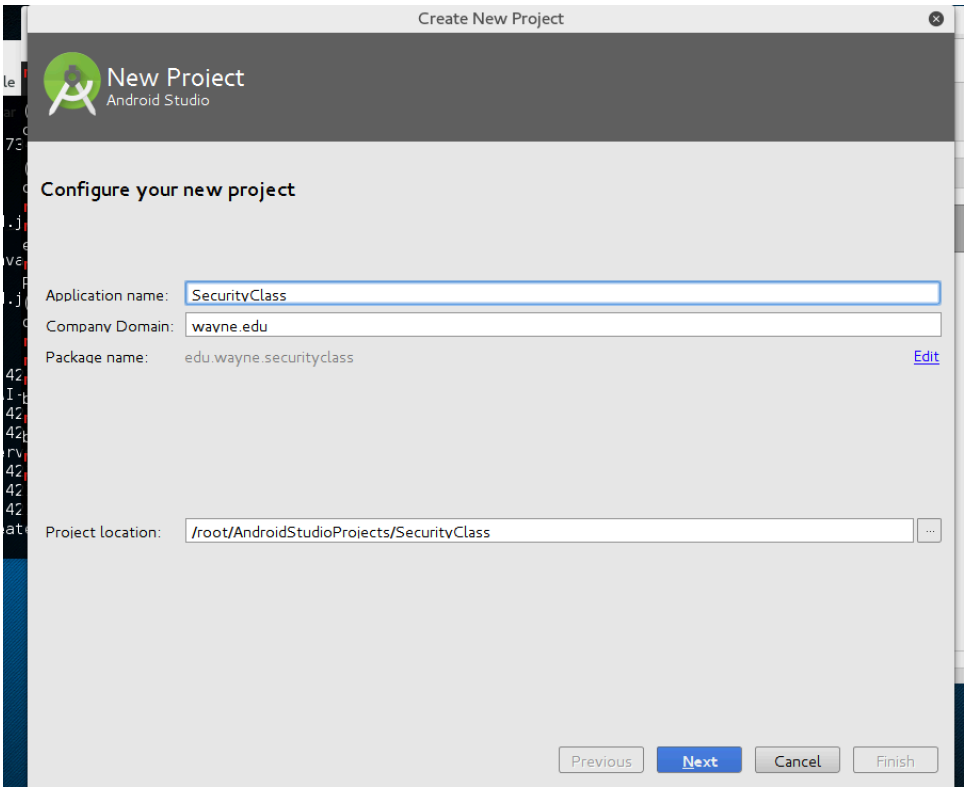
$ cd /root/Documents/android-studio/bin/

$ ./studio.sh &



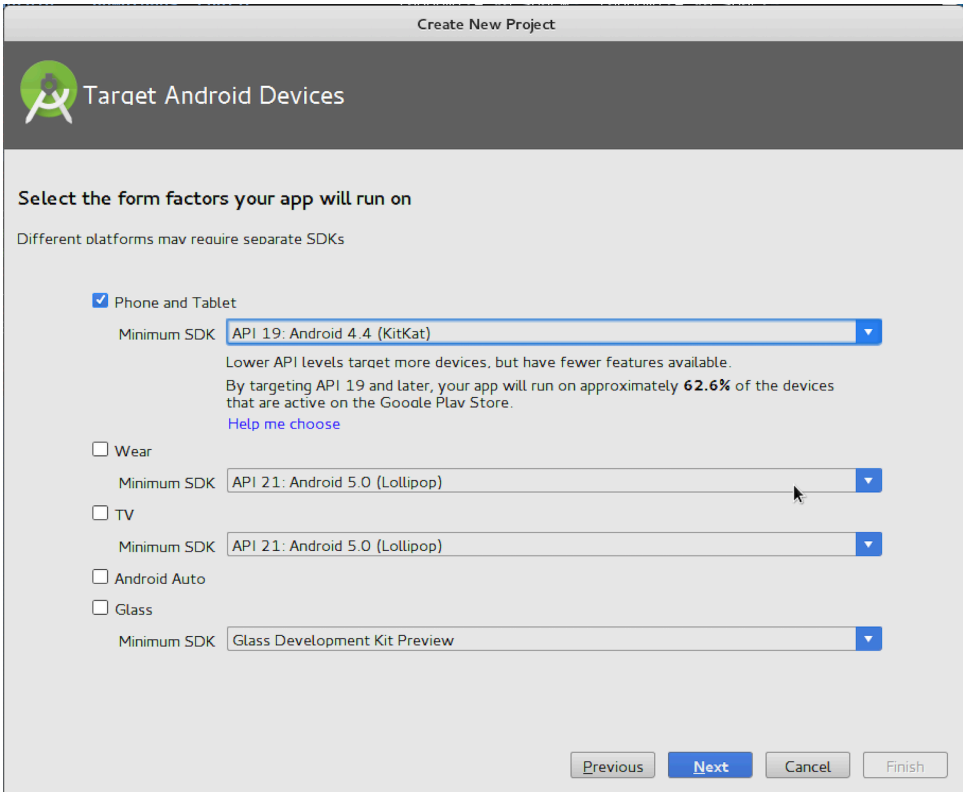
2. Start a new Android Studio Project

Click on Start a new Android Studio Project. Input the application name and the company domain. Note that the package name is the reversal of domain name and the application name. The screenshot below shows an example.

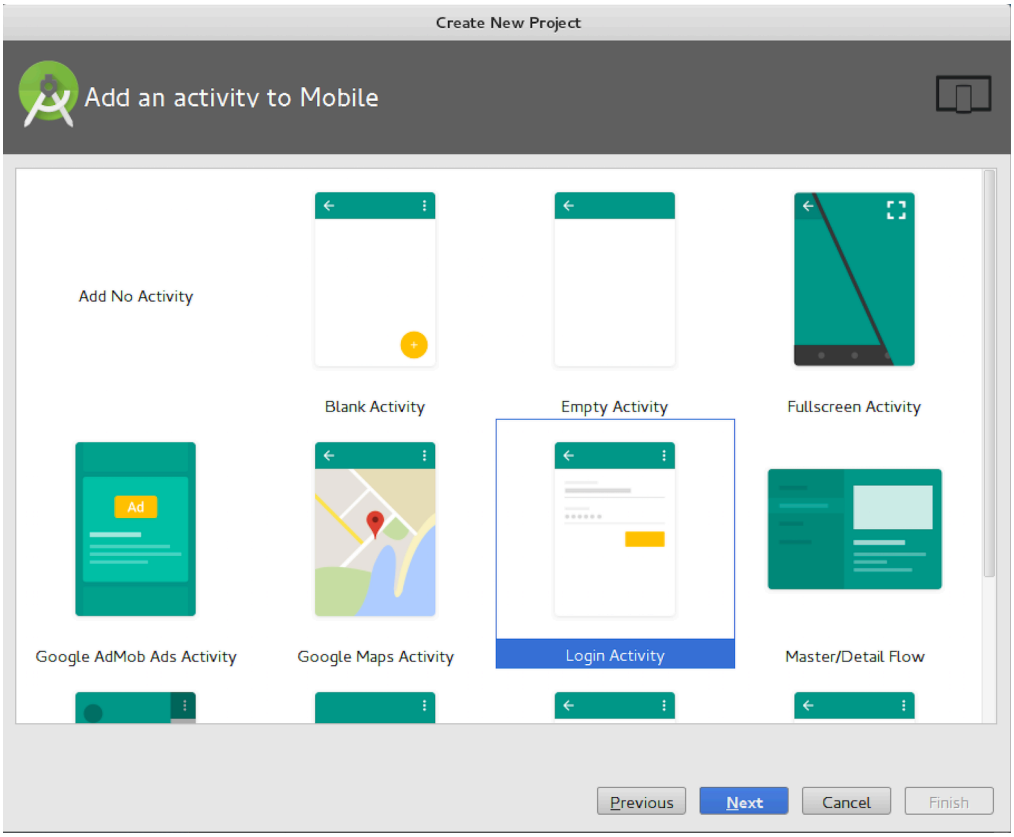


3. Select the Form Factors Your App Will Run On

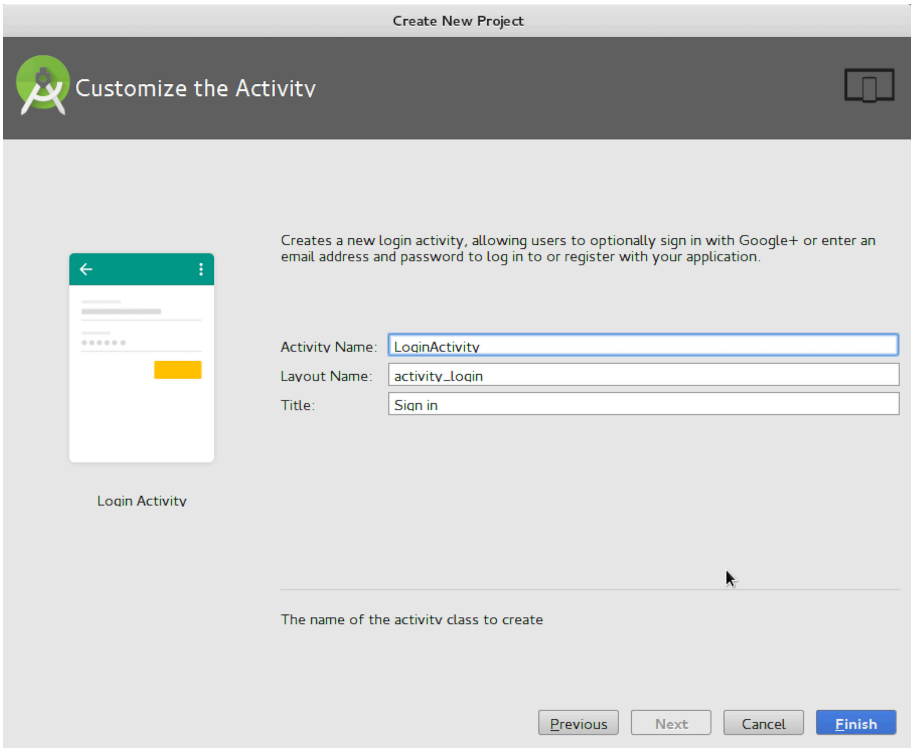
Check the Phone and Tablet box, and select the minimum SDK as API 19. Android 4.4.



4. Add the Login Activity to Mobile



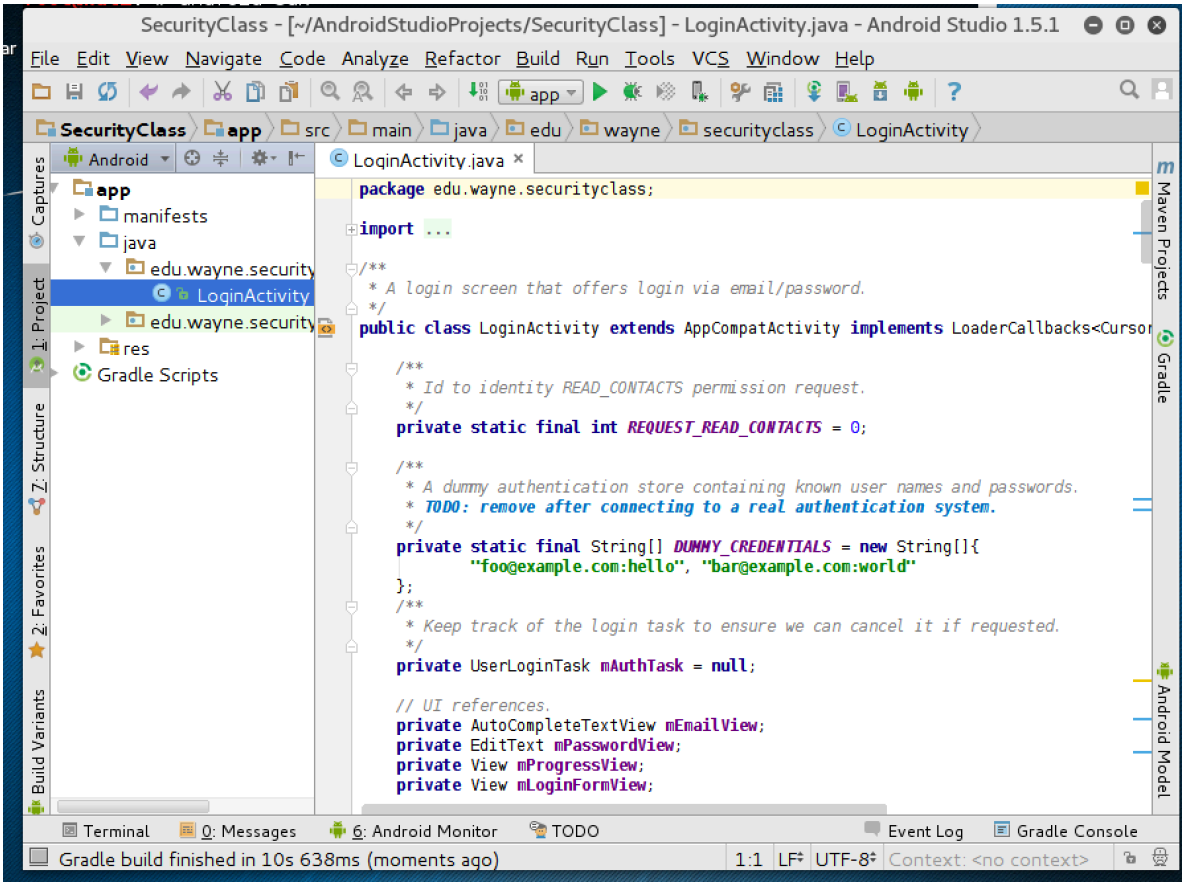
5. Customize the Activity



6. Finish Creating the Application

The screenshot in the next page shows the GUI interface of Android Studio IDE. You can see that the *LoginActivity.java* source code is the login activity you just created. The default usernames and passwords are defined in *LoginActivitly.java*.

```
/**
 * A dummy authentication store containing known user names and passwords.
 * TODO: remove after connecting to a real authentication system.
 */
private static final String[] DUMMY_CREDENTIALS = new String[]{
        "foo@example.com:hello", "bar@example.com:world"
};
```

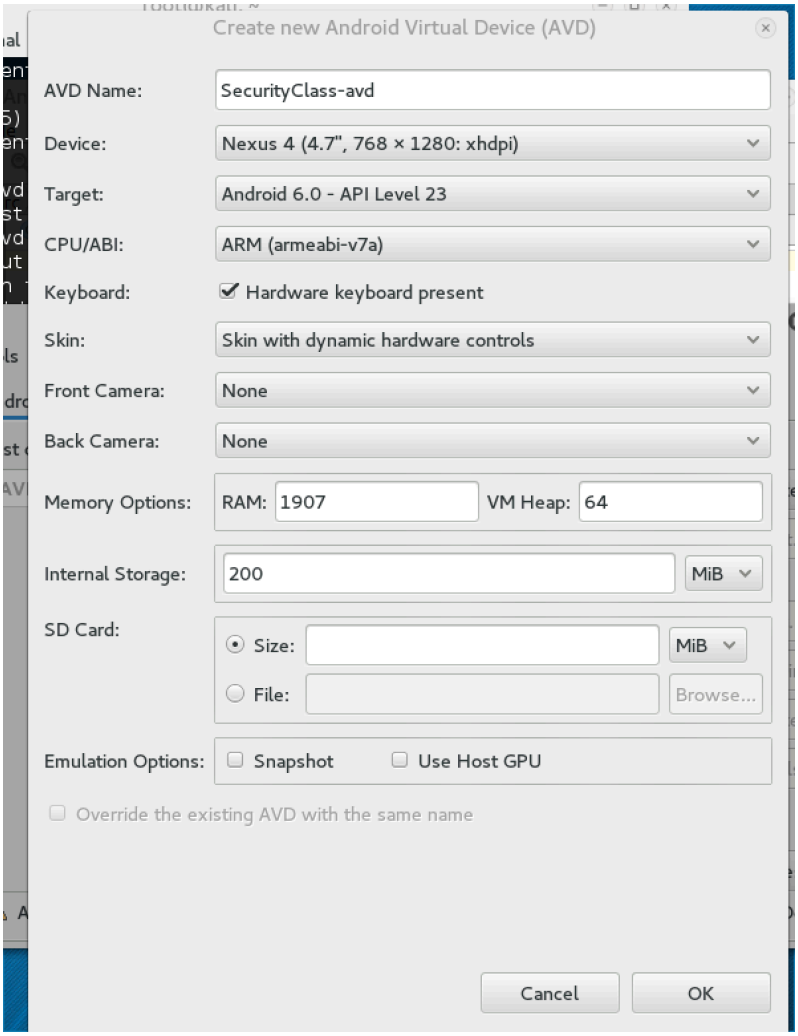

SecurityClass - [~/AndroidStudioProjects/SecurityClass] - LoginActivity.java - Android Studio 1.5.1
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SecurityClass app src main java edu wayne securityclass LoginActivity
Android
app
manifests
java
edu.wayne.security
LoginActivity
edu.wayne.security
res
Gradle Scripts
LoginActivity.java
package edu.wayne.securityclass;
import ...
/**
* A login screen that offers login via email/password.
*/
public class LoginActivity extends AppCompatActivity implements LoaderCallbacks<Cursor
/**
* Id to identity READ_CONTACTS permission request.
*/
private static final int REQUEST_READ_CONTACTS = 0;
/**
* A dummy authentication store containing known user names and passwords.
* TODO: remove after connecting to a real authentication system.
*/
private static final String[] DUMMY_CREDENTIALS = new String[]{
"foo@example.com:hello", "bar@example.com:world"
};
/**
* Keep track of the login task to ensure we can cancel it if requested.
*/
private UserLoginTask mAuthTask = null;
// UI references.
private AutoCompleteTextView mEmailView;
private EditText mPasswordView;
private View mProgressView;
private View mLoginFormView;
Terminal
0: Messages
6: Android Monitor
TODO
Event Log
Gradle Console
Gradle build finished in 10s 638ms (moments ago)
1:1
LF
UTF-8
Context: <no context>
Maven Projects
Gradle
Android Model
Captures
1: Project
7: Structure
2: Favorites
Build Variants

Creating an Android Virtual Device (AVD) with AVD Manager

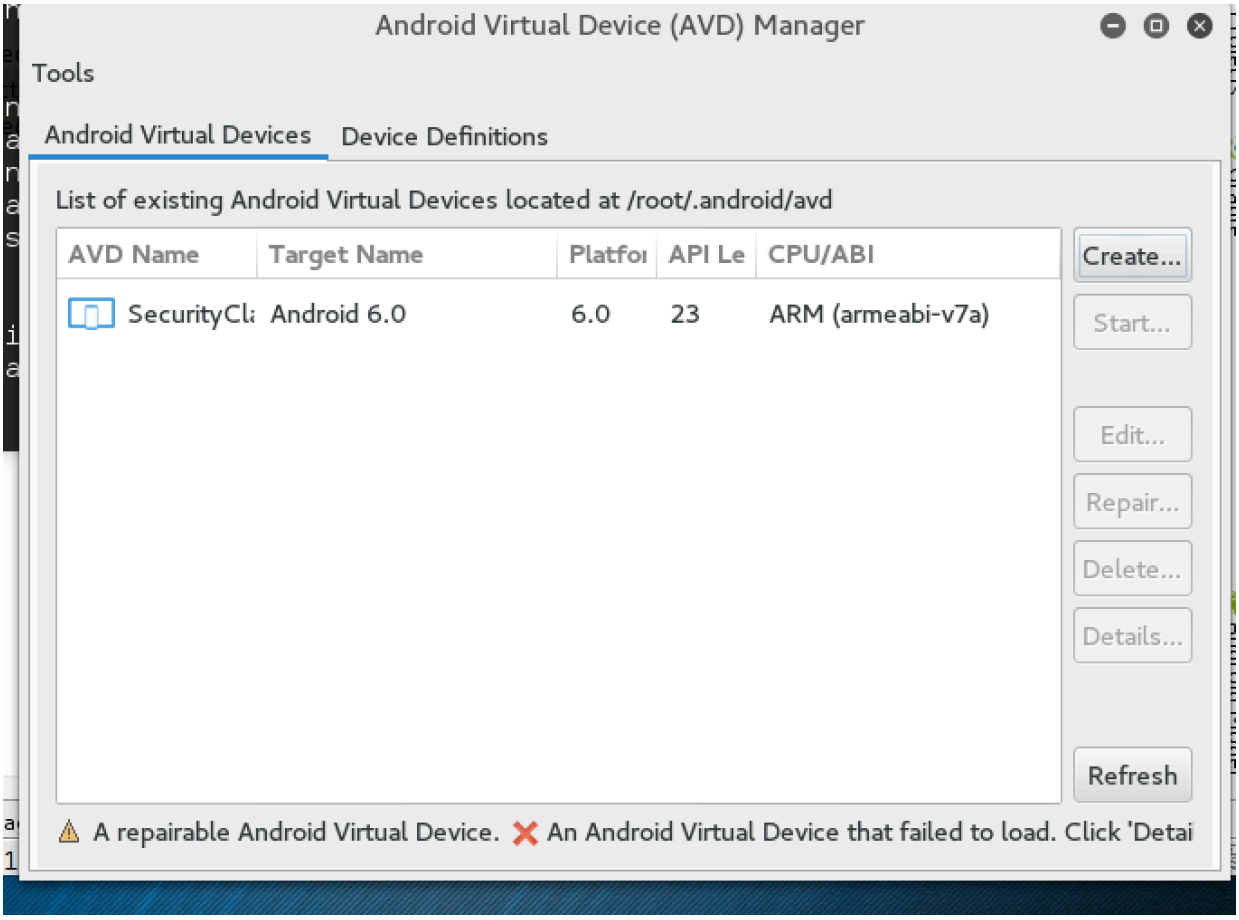
To run your application, you need to create a virtual device. If you want to run the application on your physical Android device, you can skip to next section. You can start the AVD manager by typing $ android avd in a terminal. Then, it should pop-up the AVD manager window as the screenshot below.

$ android avd

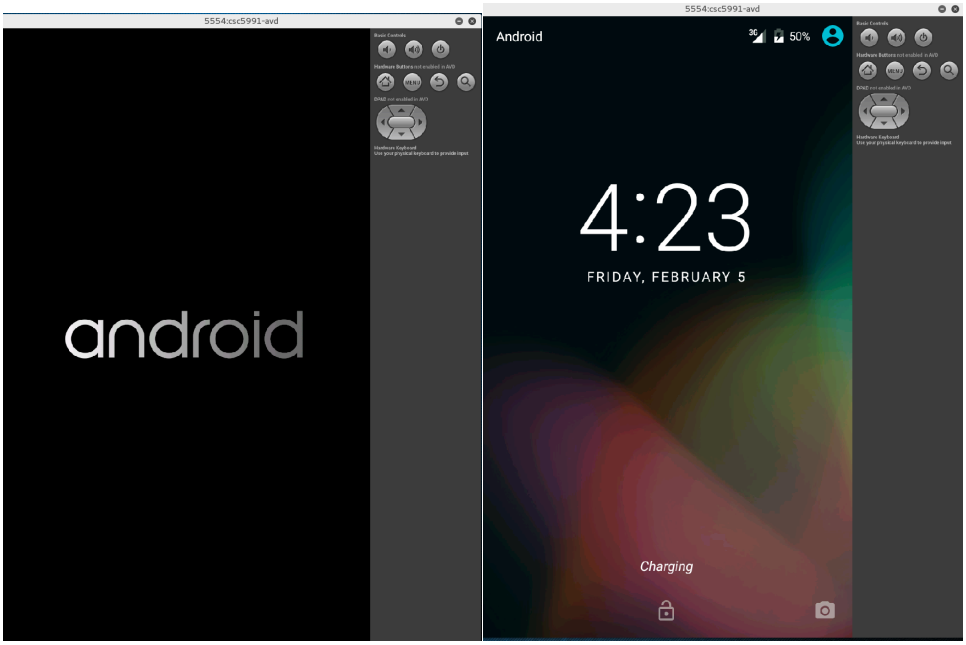
Click on “Create…” to create a new Android Virtual Device. If you create a virtual device first time, you may need to run *$ android sdk* to install the packets for CPU/ABI. The Kali VM image should have installed the required packets. The screenshot below shows creating a virtual device.



After you specify the settings for the Android virtual device, you can click “OK”. Then, the device you just created will show on the AVD manager as the screenshot below.

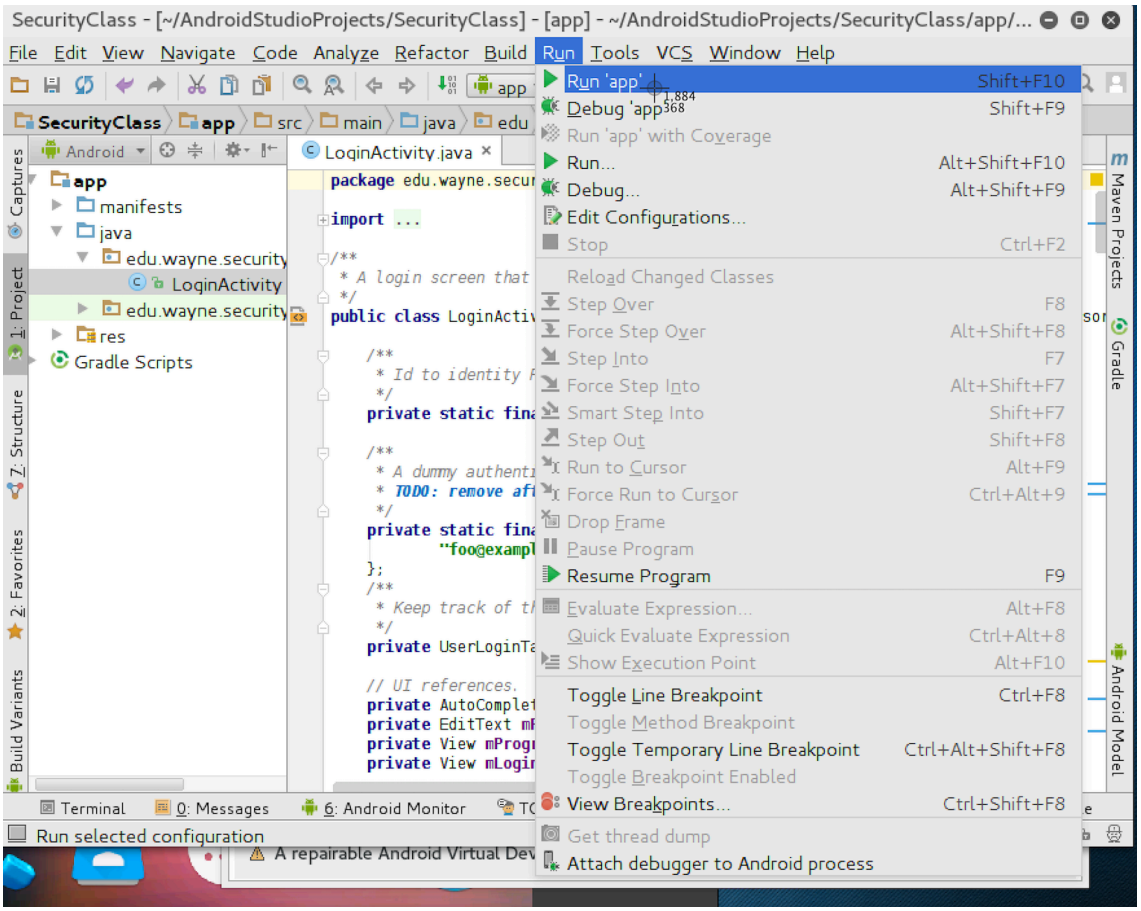


Next, you can click “Start…” to launch the virtual device using the emulator system. Note that it may take sometime to open the emulator. Be patient! The screenshots below show the Android Virtual Device in an emulator.

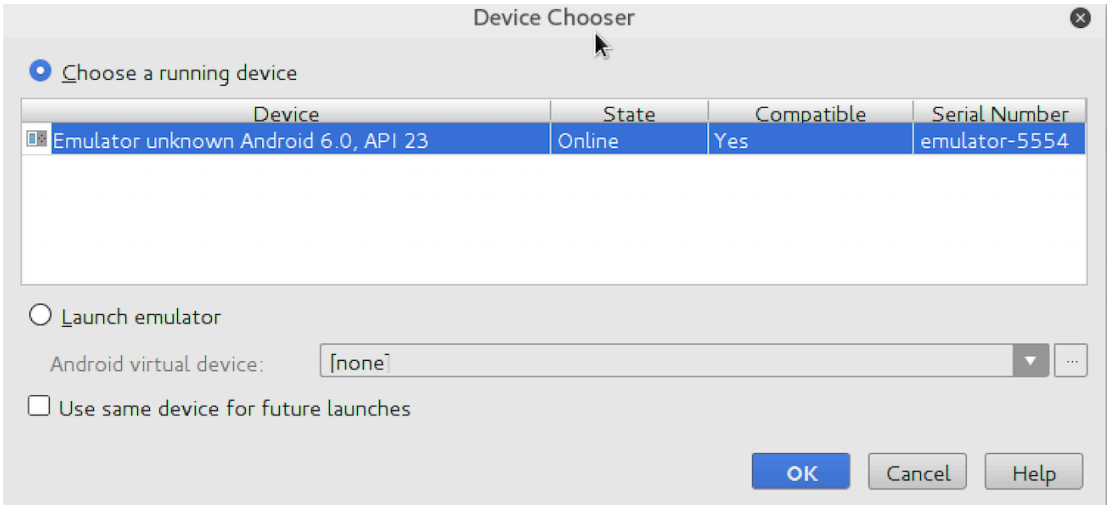


Running an Android Application

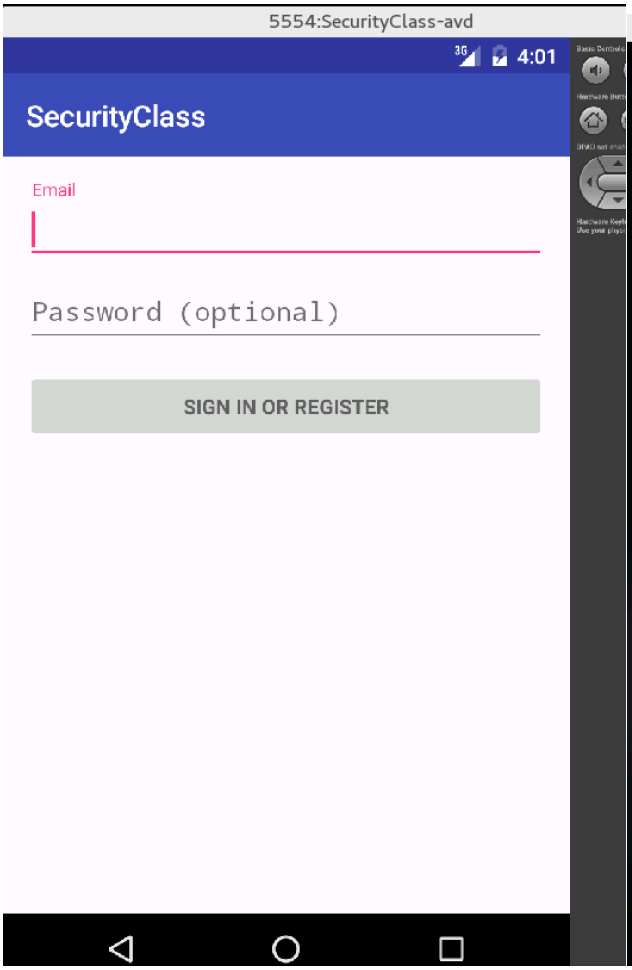
Next, you need to run the application that you just created. You can choose to run it on your own Android device or an emulator. In this lab instruction, we will show how to run it in an emulator.

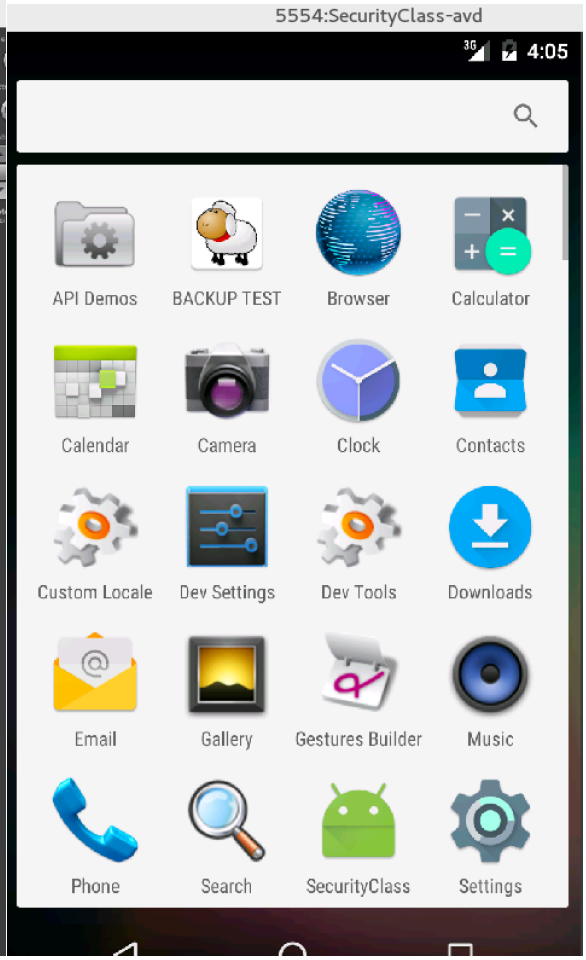


Click on “Run” tab and then select “Run “app””. Then, you will see a pop-up Window to ask you to choose a device. You can choose the running device. Click “OK”.



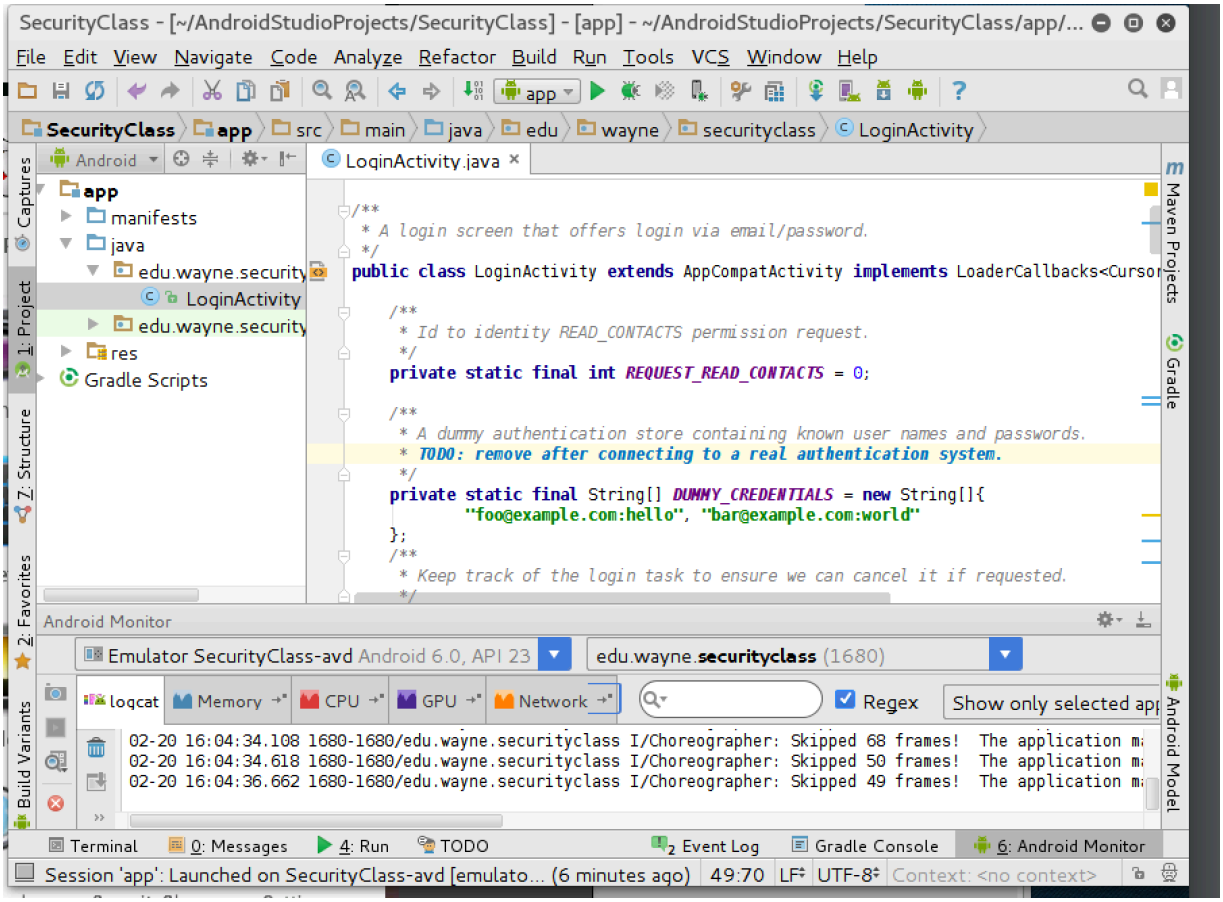
If you didn't open the emulator yet, you can mark the radio button "Launch emulator" and select the name of the AVD you just created. It is SecurityClass-avd in the example of this lab instruction. Note that it takes some time to open up an emulator, especially in a VM. Screenshots below show the SecurityClass application is running in the emulator.





You can use the default usernames and passwords to login. They are foo@example.com: hello and bar@example.com: world.

The Android Studio IDE shows the runtime information including ***Logcat*** message. We will use ***Logcat*** to show the login username and password later in this lab. The screenshot below shows the interface of Android Studio IDE whiling running the SecurityClass application.

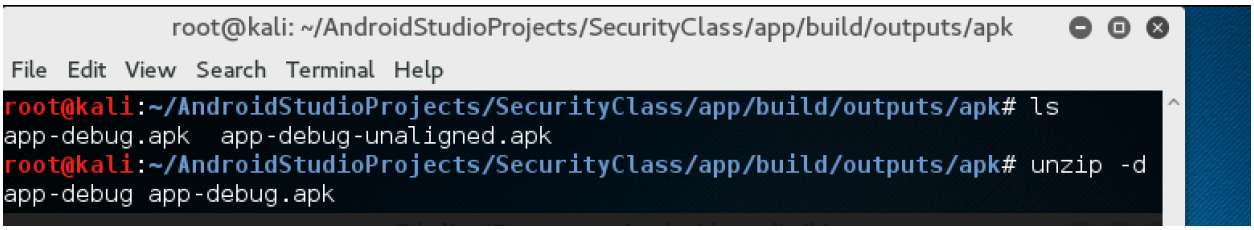


Repackage an Android Application

In this exercise, you will repackage the android application that you just created without any modification. The goal of this exercise is to help you familiar with the repackaging process.

1. Unzip Android Application apk File.

Find the apk file of the application you created (the name is SecurityClass in this lab instruction). It is at */root/AndroidStudioProjects/SecurityClass/app/build/outputs/apk/* directory. The name of the apk file is app-debug.apk. The apk file is actually a zip file to encapsulate the application code and its resource. You need to extract it to become a normal directory and 'cd' into it. Then, the current directory should contain a classes.dex file, an AndroidManifest.xml file, and additional files.



$ cd /root/AndroidStudioProjects/SecurityClass/app/build/outputs/apk

$ unzip –d app-debug app-debug.apk

$ cd app-debug

2. Convert dex to smali Format

baksmali and smali are open-source command line tools for disassembling and reassembling Dalvik bytecode to/from an intermediate representation called smali. This is the link to the source code

https://github.com/JesusFreke/smali

The smali format is an intermediate representation of the Dalvik bytecode and it is also the name or the reassembler. The baksmali.jar file disassembles the classes.dex file to a directory of smali files that are organized hierarchically according their package name. The classes.dex file contains the Dalvik bytecode. The smali files can be modified to remove and/or add in additional code into an application as long as it is consistent with the smali format. After the smali files have been modified, the smali.jar file can be used to convert the smali files back into a classes.dex file. Then additional steps are taken to repackage the application.

I have downloaded baksmali and smali tools for you under the root desktop directory of the VM image. You can download the baksmali-2.1.1.jar and the smali-2.1.1.jar from this website:

https://bitbucket.org/JesusFreke/smali/downloads

Next, you need to use the baksmali.jar file to convert the classes.dex file to a directory containing smali files that are organized hierarchically according to the package name. This will create a directory named '**out'** in the current directory that contains the smali files if you do not provide an output directory.

$ java –jar ~/Desktop/baksmali.jar classes.dex

3. Examine smali Files

Open the LoginActivity.smali file at

/root/AndoridStudio/Project/SecurityClass/app/build/outputs/apk/app-debug/out/edu/wayne/securityclass

And get familiar with the Dex instructions. The instructions in the Dalvik bytecode are explained in the links below.

https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html

http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

4. Create a New classes.dex from smali Files.

First, delete the current classes.dex file so you can create a new classes.dex file based on the smali files. Then, create a new classes.dex file from the directory of smali files using the smali.jar file. Lastly, delete the 'out' directory since you have just created a new classes.dex file.

$ cd /root/AndroidStudioProjects/SecurityClass/app/build/outputs/apk/app-debug

$ rm classes.dex

$ java –jar ~/Desktop/smali.jar –o classes.dex out

$ rm –r out

5. Encapsulate into a New apk File

Delete the directory containing the previous signature of the application's files

$ rm –r META-INF

Zip the files in the current directory to a file name of your choosing to create a zip file of the application.

*$ zip –r app-debug.zip ./**

Change the file extension of the zip file to apk if needed.

$ mv app-debug.zip app-debug.apk

6. Sign the New apk File

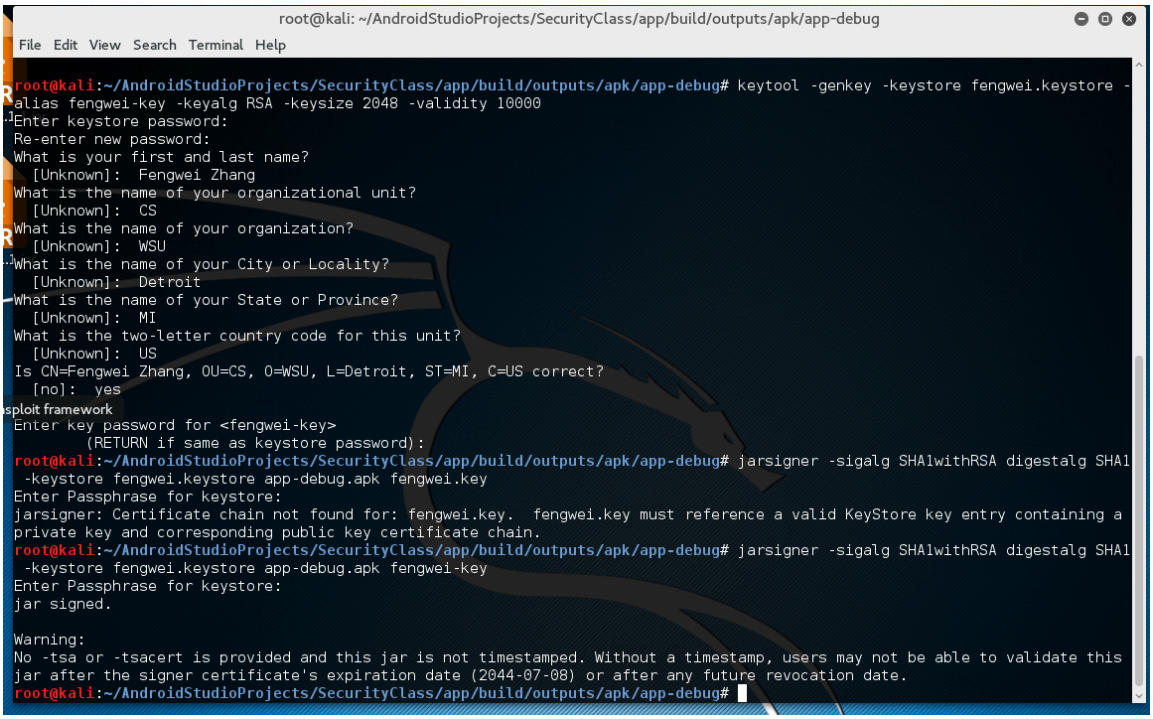
Generate a private key using *keytool*. The command below prompts you for passwords for the keystore and key, and to provide the Distinguished Name fields for your key. It then generates the keystore as a file called fengwei.keystore (Create your own keystore). The keystore contains a single key, valid for 10000 days. The alias (i.e., fengwei-key here, but create your own key) is a name that you will use later when signing your app. You can skip this step in your assignment since I have generated this key in the VM image. You also can generate another one with a different key name.

$ keytool -genkey -keystore fengwei.keystore -alias fengwei-key -keyalg RSA -keysize 2048 -validity 10000

Sign your app with your private key using *jarsigner.* The command below prompts you for passwords for the keystore and key. It then modifies the apk in-place to sign it. Note that you can sign an APK multiple times with different keys.

$ jarsigner -sigalg SHA1withRSA -digestalg SHA1 -keystore fengwei.keystore apk-debug.apk fengwei-key

The screenshot below shows the signing process.



You can use the Android Studio to sign your app. Detailed information about signing your application can be found here:

http://developer.android.com/tools/publishing/app-signing.html

7. Align the New apk Package

zipalign ensures that all uncompressed data starts with a particular byte alignment relative to the start of the file, which reduces the amount of RAM consumed by an app.

$ /root/Android/Sdk/build-tools/23.0.2/zipalign –v 4 app-debug.apk app-debug-aligned.apk

You can delete the unaligned apk and rename the aligned apk to whatever you want. The aligned application is the final product.

$ mv app-debug-aligned.apk app-debug.apk

8. Install the App with ADB

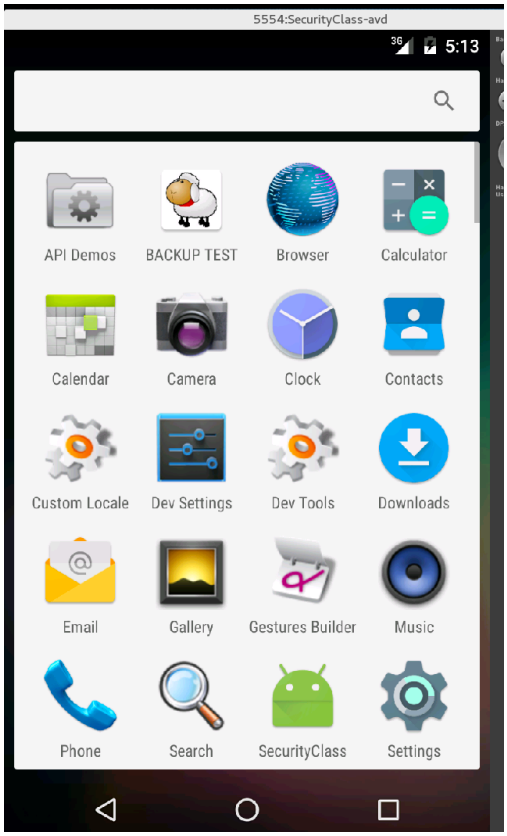
Now you can run the app by installing it via Android Debug Bridge (ADB). To use ADB, you will need to enable USB debugging on your Android device if you are not using the emulator. The emulator should have ADB enabled by default. You can then use ADB to install the app on your Android device or emulator by entering the *"adb install app-debug.apk"* command to install the app. This should install the app on your Android

device or emulator. Note that you may have installed the application, so you need to uninstall it first by executing command “adb uninstall edu.wayne.SecurityClass”

$ adb uninstall edu.wayne.SecurityClass

$ adb install app-debug.apk

You can start the emulator by AVD manager. The installed app SecurityClass will show up on the emulator as the screenshot below.



Repackage and Modify an Android Application

You will need to modify a smali file from the Login app to leak the username and password to the Android log. Please follow the instructions above for the repackaging operations as given. Any changes you make to the smali file need to be in the correct format and syntax. If you use the correct format, the modified smali files will be successfully converted into a classes.dex file. Locate the smali file where the login occurs. Then use the *android.util.Log.d(String, String)* Android API call in smali format to write both the username and password to the log. This is a static Android API call so an *android.utli.Log* object does not need to be explicitly prior to writing to the log. The username and password will be contained in a register that references a String. You can create a log tag to use as well. You can declare and initialize a String and select the register number for the log tag. Make sure that you select either a new register number or a previously used register number that is not used in the subsequent smali method that you are modifying. The first parameter to the *android.util.Log.d(String, String)* Android API call is the log tag and the second parameter is the message to write to the log. You should be able to identify the register that contains the login and password and reference them in the smali. After modifying the smali and repackaging the app, it should write the username and password to the Android log when you log in.

The *LoginActivity.java* source code will help you. You can see the Android runtime log information from Android Studio IDE. Your also can use 'adb logcat' command to view the Android log and then redirect the output to a file just prior to running the repackaged application.

Dalvik bytecode: https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html

Smali registers: https://github.com/JesusFreke/smali/wiki/Registers

Types and fields: https://github.com/JesusFreke/smali/wiki/TypesMethodsAndFields

Extra Credit: Download an app from Google Play and modify its code to leak the username and password.

Obfuscate an Android Application

The ProGuard is a built-in tool in SDK that shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names. The result is a smaller sized apk file that is more difficult to reverse engineer.

Understand the ProGuard tool, and use it to protect your application.

http://developer.android.com/tools/help/proguard.html

Packing is used to obfuscate the code of a program. It is typically used to protect the executable from reverse engineering. Find a commercial packer or Android obfuscation tool, and then pack the apk file you created. Then use the baksmali and smali tools to see if you can repackage an Android application.

Papers about unpacking Android applications

AppSpear: http://link.springer.com/chapter/10.1007%2F978-3-319-26362-5_17

DexHunter: http://link.springer.com/chapter/10.1007%2F978-3-319-24177-7_15

Assignments for Lab 5

1. Read the lab instructions above and finish all the tasks.
2. Turn in the file name and entire smali method that you modified to write the username and password to the log from the Login App.
3. Turn in a screenshot of the captured username and password
4. Describe the process to obfuscate an Android Application
    a. What tools did you use?
    b. Can you still repackage the application using baksmali or smali tool? Justify your answer.

Happy Hacking!