# Injecting Blockchain Calls with the Soot Framework and APKTool Documentation

## Contents

# 1 Terminology

| Term | Definition |
|---|---|
| compiler | A program that translates statements written in a source programming language and into machine language, object code or assembly. |
| decompiler | A program that translates machine language, object code or assembly into a high level language such Java. |
| bytecode | A low-level representation of program code that has been compiled. It can closely resemble assembly language. |
| APK | The Android Package Kit is used to distribute and for the subsequent execution of an Android application. It is similar to the exe format in Microsoft Windows. |
| code injection | The process of injecting statements into an application at a specific location without disturbing the flow of the application code. |
| soot | A compiler framework that is able to decompile and compile Java code with the capability of analysing and instrumenting Java code. |
| instrumentation | Refers to the modification and analysis of a programming language through the use of compiler technology. |
| jimple | An intermediate representation of Java code that Soot generates as output. |
| APKTool | A compiler framework that is able to simply decompile and compile Java code. |
| smali | An intermediate representation of Java code that APKTool generates as output. |
| blockchain | A peer-to-peer network that allows for the sharing of data among a vast number of peers [1]. All data stored on the blockchain is immutable. |
| Ethereum blockchain | A blockchain environment that allows the use of smart contracts. |
| smart contract | A contract with written rules and terms allowing for controlling the storage, sharing, and modification of data. |
| Ganache | A tool used for creating an Ethereum blockchain environment. |
| solidity | A smart contract object-oriented programming language that was developed by Ethereum. |
| Remix | Ethereum's tool that helps developers program smart contracts. It enables smart contract developers to connect and push smart contracts to the Ethereum blockchain. |
| DApps | This refers to the decentralized, resilient, transparent, and incentivised applications that reside on blockchain infrastructures. These applications are supposedly less prone to errors. |

# 2 Purpose of the documentation

This documentation is about how to set up the APKTool and Soot framework compiler environment. Also, it is about how to inject blockchain calls into an Android application using APKTool and the Soot framework. We discuss how to set up the Ethereum private blockchain environment.

Figure 1 illustrates the process of injecting a blockchain call into an Android application. Remix was used to connect to Ganache. Ganache is the private blockchain environment that contains our smart contract. Remix was used to connect to Ganache and send the smart contract to Ganache. Then Soot/APKTool was utilized to inject the blockchain call into the test application. The Android test application already had the API installed to eliminate some complexity. The Android application when installed and run on the Android phone emulator sends the data to Ganache through the Web3j API. This might appear to be simple, but in reality it is not and becomes very time consuming.

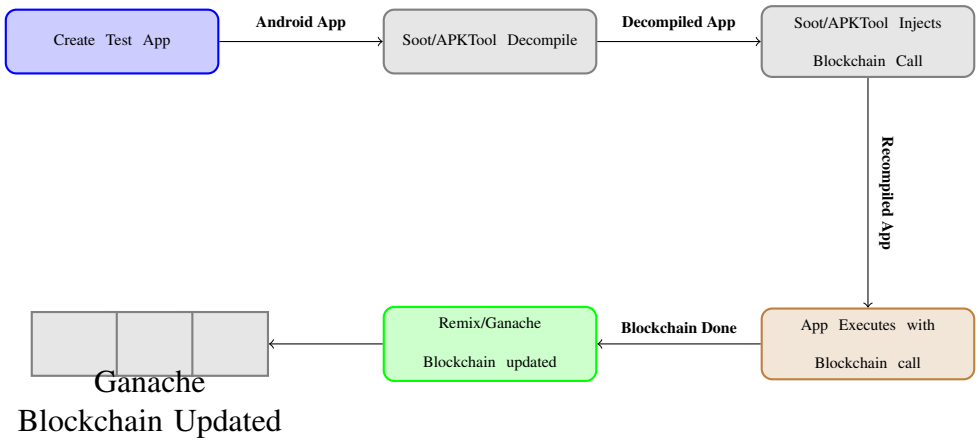


Figure 1: Overview of code injection

# 3 Environment Used

This documentation is intended to be followed using Linux operating systems. We used Ubuntu 20.0.4 LTS because of the ease of use and reliability that the distribution provides. Also, we have installed openjdk version 11.0.7. This is the version of Java that is compatible with the Soot framework.

# 4 Setting up Ganache

Ganache is a tool that allows the generation of the private blockchain environment. Ganache allows an effortless way of specifying the blockchain accounts and other settings for the private blockchain environment. Without Ganache, it would be nearly impossible to set up a private Ethereum blockchain environment because a command-line tool called geth would be required to set up a private blockchain. The geth command-line tool is very tricky to set up and to set the correct input parameters.

## 4.1 Installation

Ganache can be downloaded from `https://www.trufflesuite.com/ganache`.

## 4.2 Running Ganache

To run Ganache, navigate to the folder where the AppImage file is at. An AppImage file is a compressed image with all the dependencies and libraries needed to run the desired software.

The first step is to make the AppImage an executable. This can be accomplished in Linux by making sure to make the file an executable.

To make a file executable in Linux:

1. right click the file
2. select properties
3. Navigate to the permissions and then select the check-box that says *Allow executing file as program*

Now you should be able to click on the file to run Ganache. A similar screen, as shown in Figure 2 will be presented.

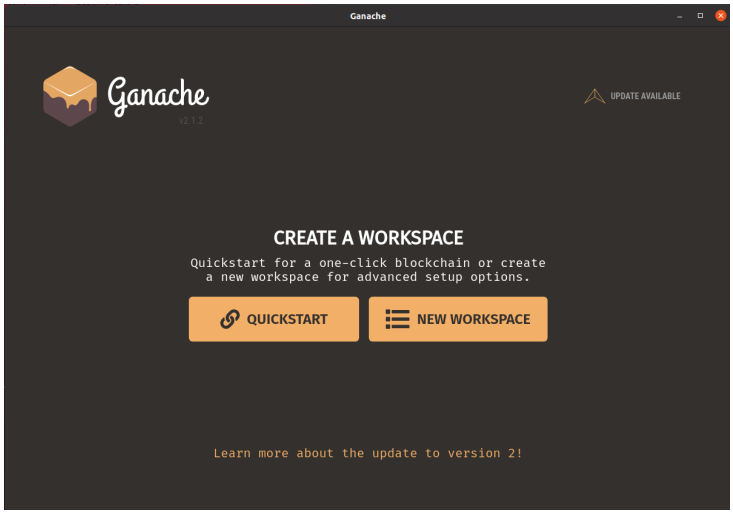


Figure 2: Ganache Start Screen

## 4.3 Setting up the private Ethereum blockchain environment with Ganache

1. Click on the QUICKSTART button. A similar screen to Figure 3 should be displayed.

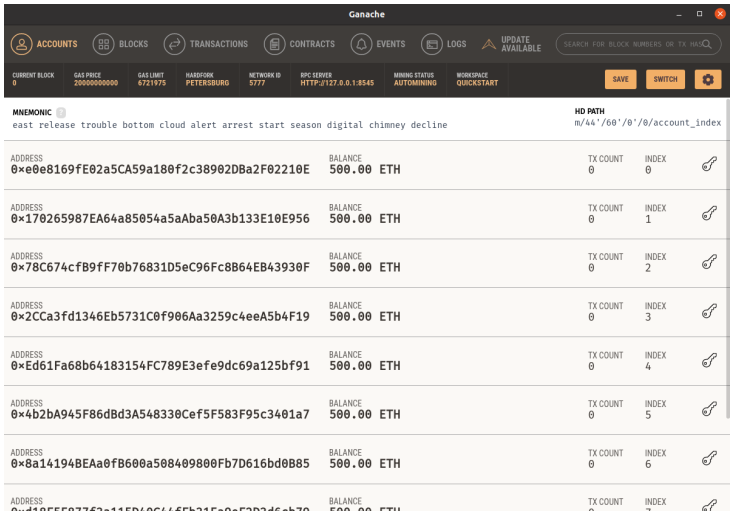


Figure 3: Ganache Quickstart Screen

2. Next, click on the  button, then click on the Server tab. Be sure to duplicate the settings as displayed in Figure 4.

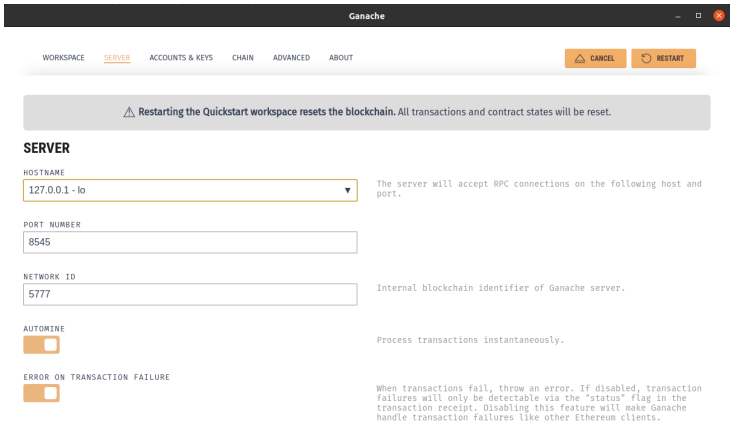


Figure 4: Ganache Settings Screen

3. Click on the settings tab and select the *Accounts & Keys tab*.

4. Copy and paste the phrase: *east release trouble bottom cloud alert arrest start season digital chimney decline* into the box next to where it says *Enter the Mnemonic you wish to use*.

5. Click on the *RESTART* button to restart Ganache. (This step is required, so that there is no longer a need to change any address that was set in the blockchain injection code that is provided in this documentation. Also, the opening and closing of Ganache application should change none of the settings that were entered.)

# 5 Remix

Remix is Ethereum's graphical user interface that allows developers to program their smart contracts. It also allows them to connect to the private blockchain environment. This section discusses how to set up the remix Environment.

## 5.1 Setting up the Ethereum Remix Integrated Development Environment (IDE)

Setting up Remix IDE is a crucial first step. The solidity programming language semantics and how to code using the solidity programming language will not be

discussed in this documentation. There are plenty of useful resources that explain the programming aspect. The main focus is on how to set up and get started with the Remix IDE.

Here are the steps:

1. Open either your Chrome or Firefox web browser and navigating to `http://remix.ethereum.org/`. The image shown in Figure 5 will then be displayed.

2. Click on the button *Solidity* located under Environments.

   Notice on the left-hand side that the ribbon has included more click-able options.

3. Set the compiler version to *0.5.15+commit.6a57276f*.

   The compiler is set to 0.5.15 version because it is specific to the syntax that the solidity code is referencing.

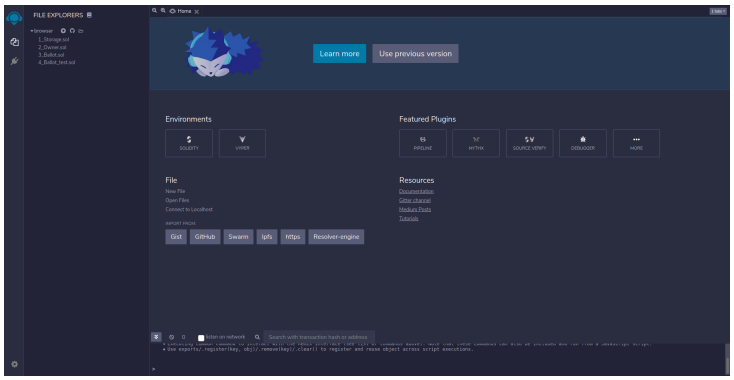


Figure 5: Main Screen

4. Click on the  icon located on the left side ribbon. Then the screen, as shown in Figure 6, will be presented. Also, the same example files should be present, but do not worry if you don't have these files. We will not be using them.

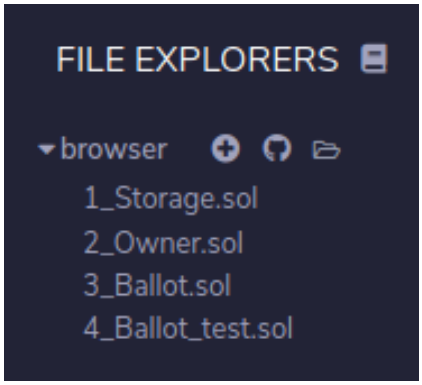


Figure 6: File Explorer Screen

## 5.2 Creating our Smart Contract using Remix

It is now time to create our smart contract. Here are the steps:

1. create a new file by clicking on the button located above the file named Storage.sol.
2. name your file "**Hello.sol**" and click OK.
3. Copy and paste the code from Listing 1 into the empty file that was created.
4. Hit the buttons *ctrl* + *s* simultaneously to save the file.

   This should trigger the compiler to compile automatically.
   If not just be sure to compile the file manually. If compiled correctly, you would see on the left side ribbon.

```
pragma solidity >=0.5.15;

contract ApplicationContract{

  string Name ="";

  function SetName(string memory statedata) public{
    Name = statedata;
  }

  function Hello()public view returns(string memory){
```

```
12      return Name;
13    }
14  }
```

Listing 1: Hello World Solidity Code

## 5.3 Connecting Remix to the Ethereum Blockchain Environment

Now it is time to connect Remix to our private Ethereum blockchain environment. This is accomplished by:

1. Click on the button which should provide a screen similar to Figure 7.

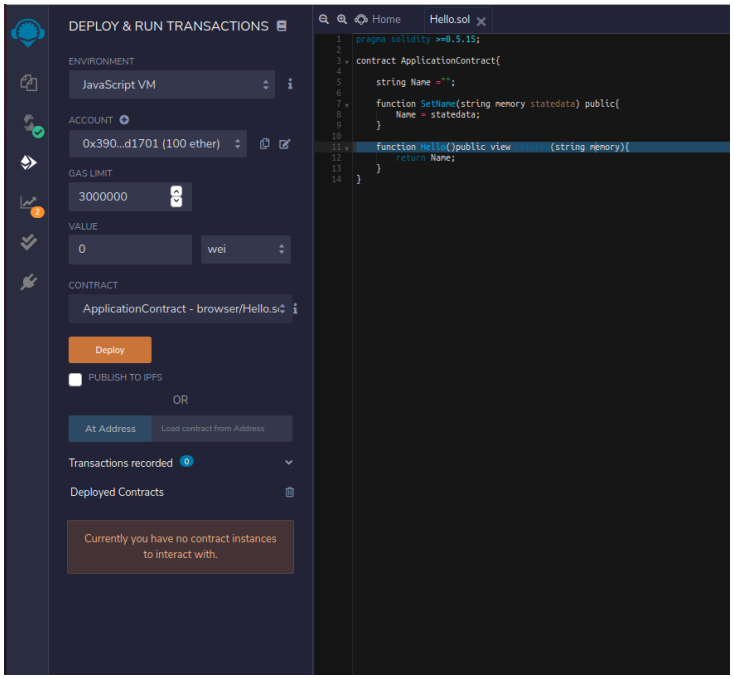


Figure 7: Remix Deploy Screen

2. Select **Web3 Provider** from the Environment drop-down menu.

3. Then a screen as displayed in Figure 8 will appear. Click the OK button.

Remix should now be connected.
If you get the error, *Cannot get account list: Error: Invalid JSON RPC response: ''*.
The error is saying it can not connect to the private blockchain.
The solution to this problem is to make sure Ganache is running and that Gnache is set to your local-host IP.

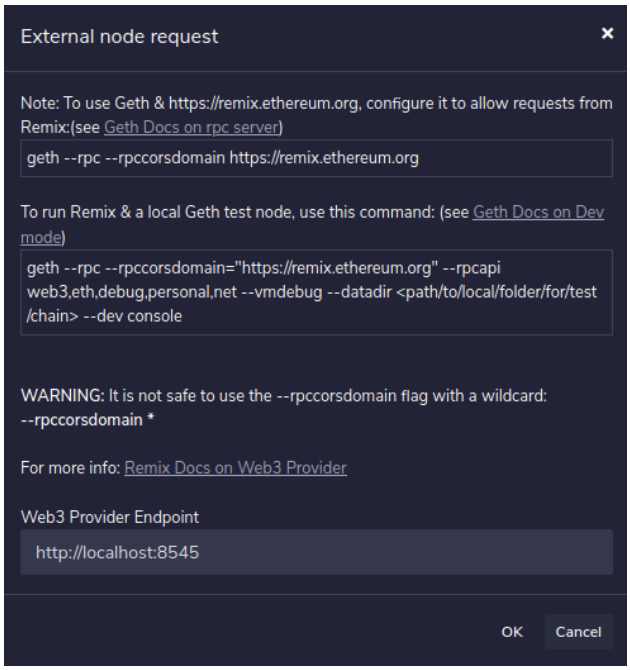


Figure 8: Remix Deploy Options

## 5.4 Deploying the smart contract using Remix

Now lets deploy our smart contract to the Ethereum Private blockchain.

Deploy the smart contract by clicking on the 

button.
Once completed, a screen similar to Figure 9 will be displayed.

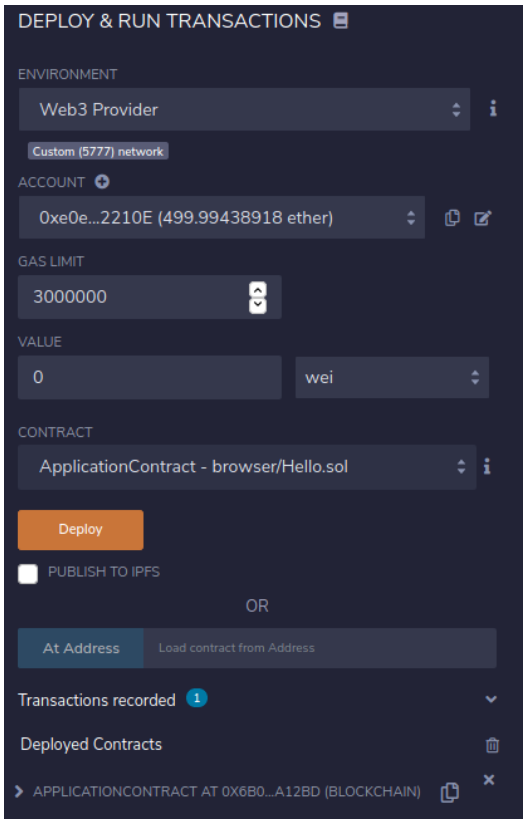


Figure 9: Remix Deployed Smart Contract

If the button is pressed, a screen similar to Figure 10 will be displayed. It is possible to directly interact with the smart contract from Remix, but it is not encouraged to do this. Make sure not to enter in any data because we want to test that the code injection works via Ganache and not via the Remix interface. If any data was entered prematurely, there would be data present on the private Ethereum blockchain.

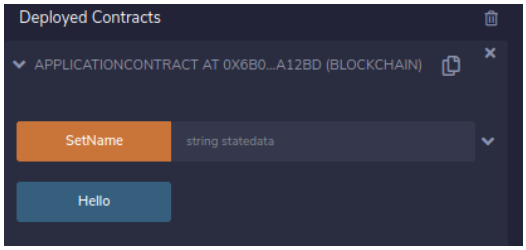


Figure 10: Deployed Hello Smart Contract

# 6 Android Application Setup

Here, we will begin by discussing where you can find the information on how to install Android Studio on your operating system. This is not a definitive guide, and the process might vary depending on if you are using Linux, Mac, or Windows.
Since we are using Ubuntu, a Linux distro, we will be focusing mainly on instructions related to the Ubuntu setup. Next, we will discuss setting up Android Studio to match our environment setup. Then we will conclude the section by discussing the project setup.

## 6.1 Installing Android Studio

To install Android Studio:

1. Open your web browser and navigate to `https://developer.android.com/studio`.
2. Click on the download button and accept the agreement.

   Wait for the download to complete.
3. Now open up the terminal by pressing *ctrl+alt+t* at the same time.

4. Next, issue a *ls* command to ensure that the .tar.gz file exists.

5. Type in *tar xzf [filename].tar.gz*. The command should be similar to Listing 2. Note, filename is the name of the downloaded file.

```
tar xzf android-studio-ide-193.6514223-linux.tar.gz
```

Listing 2: Extracting .tar.gz file

6. Next, change directories by issuing the command shown in Listing 3.

```
cd android-studio/bin
```

Listing 3: Change directory to Android Folder

7. From here you can run android studio by issuing the command shown in Listing 4.

```
./studio.sh
```

Listing 4: Running Android Studio

## 6.2 Setting up the Android Studio Environment for using the Soot framework

Now lets begin setting up the Android Studio environment. First we will discuss how to setup the Android AVD Manager. Then we will discuss how to setup the Android SDK manager.

### 6.2.1 Android AVD Manager Setup

1. Start up Android studio and navigate to the start screen, as shown in Figure 11.

2. Click on $Configure \rightarrow AVDManager$.

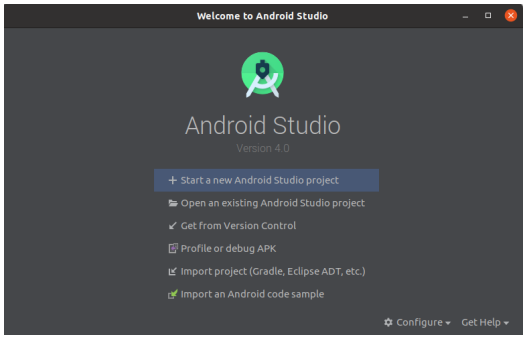


Figure 11: Welcome Screen

3. Click on **Create Virtual Device...**.
4. Select Pixel 2 as shown in Figure 12, click Next, click next again, and then click the Finish button.

   Be sure to click on the red x to close the window.

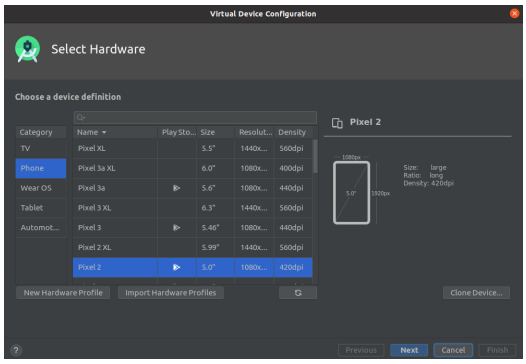


Figure 12: Phone Device

## Android SDK Manager Setup

1. Click on $Configure \rightarrow SDKManager$.
2. Select *Android Pie 9.0 API level 28*.
3. Click on the button to download the SDK.
4. Select *Apply* and then the *OK* button.

## 6.3 Android Project Setup

Not it is time to create a new Android Studio project. This is accomplished by:

1. going to the welcome screen, as displayed in Figure 13.

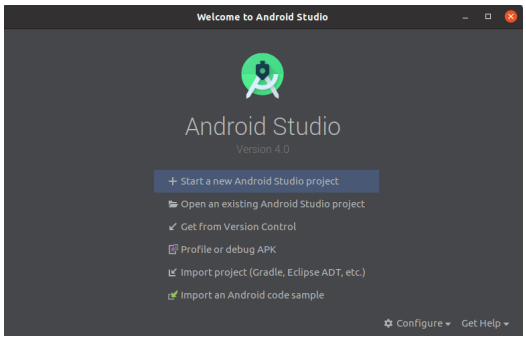


Figure 13: Welcome Screen

2. Click on the *Start a new Android Studio Project* button.
3. Click on EmptyActivity, as displayed in Figure 14.

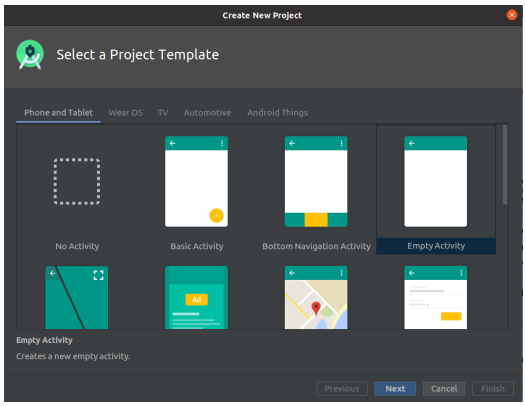


Figure 14: Empty Activity

4. Click the Next button and have the same configurations, as shown in Figure 15.

   Note, you will not have the same save location, which is OK. Just make sure you save it to a location that you will remember for future use.

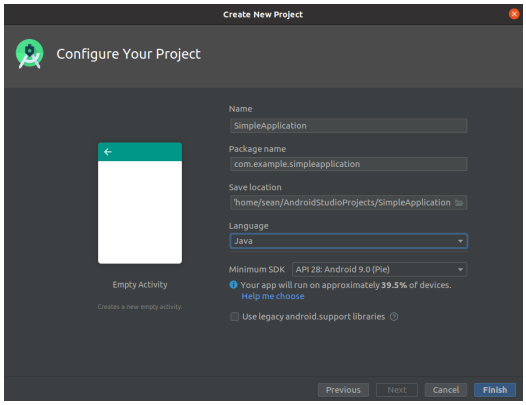


Figure 15: Final Configuration Settings

### 6.3.1 Gradle Setup

Now lets setup the Gradle Scripts for the Android Studio project. The script is used to help integrate the required libraries into our Android application without having to worry about downloading the wrong library.
Setup the gradle scripts by:

1. Navigate to *Gradle Scripts* $\rightarrow$ $build.gradle(Module : app)$ which is in your Project File explorer window (Figure 16).

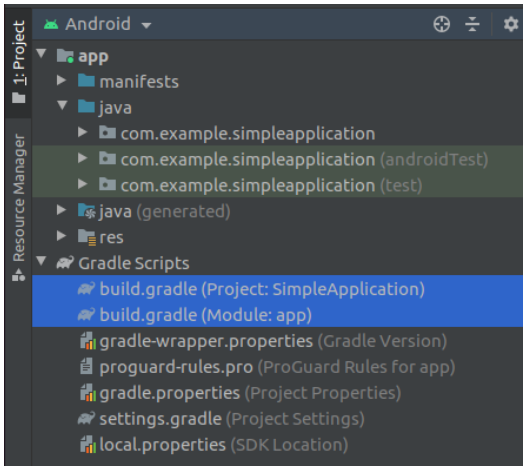


Figure 16: Project Explorer Window

2. Copy the contents from Listing 5 and paste it into your own *build.gradle(Module:app)* file.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    buildToolsVersion "28.0.3"

    defaultConfig {
        applicationId "com.example.simpleapplication"
        minSdkVersion 28
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.
    AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-
    android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility = 1.8
        targetCompatibility = 1.8
    }

}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:
    constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit
    :1.1.1'
    androidTestImplementation 'androidx.test.espresso:
    espresso-core:3.2.0'
//    implementation fileTree(dir: 'libs', include: ['*.aar
```

```
    ', '*.jar'], exclude: [])
    implementation('org.web3j:core:4.5.6')
    implementation('org.slf4j:slf4j-simple:1.7.29')
}
```

Listing 5: build.gradle(Module:app)

3. Navigate to your *build.gradle(Project:SimpleApplication)* file and paste the contents from Listing 6 into your *build.gradle(Project:SimpleApplication)* file.

```
// Top-level build file where you can add configuration
    options common to all sub-projects/modules.

buildscript {

    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.6.2'


        // NOTE: Do not place your application dependencies
    here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Listing 6: build.gradle(Project:SimpleApplication)

4. Re-sync your Gradle files by clicking on the *Sync Now* button that appears on the top of your file, as shown in Figure 17.

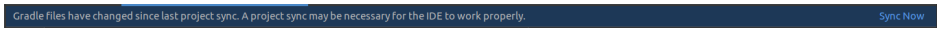


Figure 17: Gradle Sync Now

### 6.3.2 Android Manifest File Setup

The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play. The Android Manifest file includes the activities, services, broadcast receivers, and content providers. Also, it includes the permissions that the Android application has. It is required to set some permissions to allow our private blockchain to communicate with our Android application and allows for the sending of data to the private blockchain.
To set up our Manifest file we will need to:

1. Navigate to the file explorer window and go to $App \rightarrow Manifest \rightarrow AndroidManifest.xml$.

2. Open the file and copy the contents from Listing 7 and paste it into your *AndroidManifest.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
     package="com.example.simpleapplication">
     <uses-permission android:name="android.permission.
    INTERNET" />
     <uses-permission android:name="android.permission.
    ACCESS_COARSE_LOCATION" />
     <uses-permission android:name="android.permission.
    ACCESS_FINE_LOCATION" />
     <uses-permission android:name="android.permission.
    ACCESS_NETWORK_STATE" />
     <uses-permission android:name="android.permission.
    ACCESS_WIFI_STATE" />
     <application
         android:usesCleartextTraffic="true"
         android:allowBackup="true"
         android:icon="@mipmap/ic_launcher"
         android:label="@string/app_name"
         android:roundIcon="@mipmap/ic_launcher_round"
         android:supportsRtl="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>
    </application>

</manifest>
```

Listing 7: Android Manifest)

### 6.3.3 MainActivity File Setup

To alleviate some of the complexity of the code injection we are providing you with the MainActivity code. This will ensure that you can successfully inject the code using APKTool and the Soot framework.
To set up the MainActivity.java file, follow the next steps:

1. Navigate and open our java file by going to
   $App \rightarrow Java \rightarrow com.exmaple.simpleapplication \rightarrow MainActivity.java$.

2. Copy and paste the contents you see in Listing 8 into your MainActivity.java file.
   Then click on the *Save* button, located under the *File* tab.

```
package com.example.simpleapplication;

import android.os.Bundle;
import android.util.Log;

import androidx.appcompat.app.AppCompatActivity;

import org.web3j.crypto.Credentials;
import org.web3j.protocol.Web3j;
import org.web3j.protocol.http.HttpService;
import org.web3j.tx.gas.ContractGasProvider;

import java.math.BigInteger;
```

```
14
15 public class MainActivity extends AppCompatActivity {
16     private Web3j web3j;
17     Hello contract = ContractInit();
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_main);
23     }
24     public Hello ContractInit(){
25         web3j = Web3j.build(new HttpService("http
    ://10.0.2.2:8545"));
26
27         ContractGasProvider contractGasProvider = new
    ContractGasProvider() {
28             @Override
29             public BigInteger getGasPrice(String
    contractFunc) {
30                 return BigInteger.valueOf(3000000);
31             }
32             @Override
33             public BigInteger getGasPrice() {
34                 return BigInteger.valueOf(3000000);
35             }
36             @Override
37             public BigInteger getGasLimit(String
    contractFunc) {
38                 return BigInteger.valueOf(3000000);
39             }
40             @Override
41             public BigInteger getGasLimit() {
42                 return BigInteger.valueOf(3000000);
43             }
44         };
45
46
47         Credentials credentials = Credentials.create("0
    x2d2c66151300672e8ec9b8c7315ef744115ff7b82f1e0b9a8bbf39452a3add28
    ");
48         Log.d("Address",credentials.getAddress());
49         Hello contract = Hello.load("0
    x6B0c9E90a645e037949876dCf23523f7185A12bd",web3j,
    credentials, contractGasProvider);
50
```

```
51          return contract;
52      }
53  }
```

Listing 8: MainActivity.java file

### 6.3.4 Generating and including the Java wrapper class in the Android Studio project

Now lets setup the wrapper Java class that will allow the Android application to connect to interact with the smart contract on the private blockchain.
To setup the Java file be sure to:

1. Go to Remix and click on the  .
2. Click on the button  .
3. Then go into the terminal window, type in *sudo apt-get install mousepad*, and then hit enter.
4. Type in *sudo mousepad Hello.abi*, hit enter, and do a *ctrl + v* to paste what you copied from Remix and save the file.
5. Copy the bytecode by going to Remix and click on  .
6. Go back to your terminal window and type in *sudo mousepad Hello.bin*.
7. Hit *ctrl + v* to paste what you copied from Remix and save the file.
8. Type in *web3j solidity generate –javaTypes -b Hello.bin -a Hello.abi -o . -p Example* into our terminal window at the same location, we have our abi and bin file.

   The files can be seen by typing in the *ls* command. This command will create an Example folder with our Hello.java file inside.
9. In the terminal window type in *pwd* to see what current working directory we are in.

10. Navigate to our file explorer and go to the location where the current working directory was set to.

    Be sure to go into the folder and select the Hello.java file and hit *ctrl + c* to copy the file.

11. Go back to Android Studio and go to $app \rightarrow Java \rightarrow com.exmaple.simpleapplication$.

12. Hit *ctrl + v* to paste the Hello.java file into the *com.exmaple.simpleapplication* directory.

13. Accept all the pop-up messages.

    Now the MainActivity and Hello Java file should be nested under the *com.exmaple.simpleapplication* directory.

# 7 Blockchain Injection Using Compilers

How to inject blockchain calls using the APKTool and Soot framework will be discussed in this section. First, we will discuss some background information about the Soot compiler, then we will discuss how to perform the blockchain code injection using the Soot compiler. Then we will go over how to install and sign the APK. Finally, we will discuss APKTool.

## 7.1 Soot Compiler Framework

The Soot framework is a Java optimization tool. But the Soot tool is now used to analyse Android applications and instrument class files. The soot framework reads in Java files, Android APK files, jimple files, and jasmin files. For this article, we will focus on the use of Soot to read in Android APK files (Figure 18). Output from Soot consists of output Java/Android byte-code, jimple files, and jasmin files. Our focus is on APK file outputs.

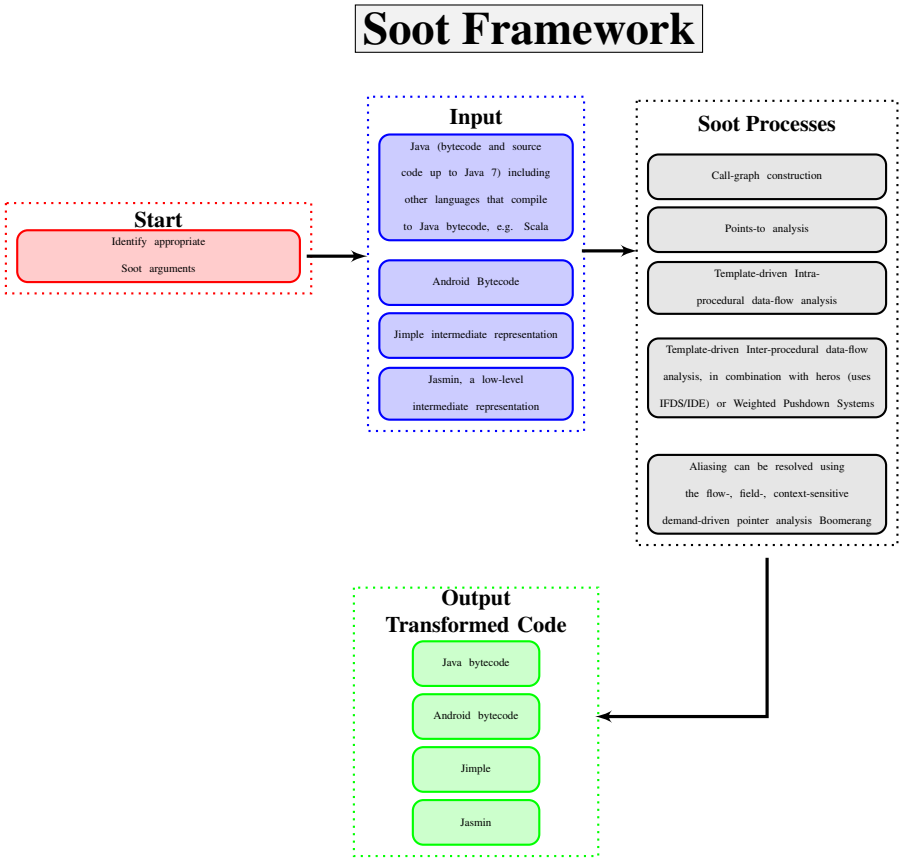


Figure 18: Soot features

## 7.2 Jimple Code

Jimple is the simplified java code format that the Soot framework uses for the constructing and deconstruction of Android applications.
Now, let us discuss the structure of what the code will look like once we inject all of our blockchain calls into our Android Application. Below in Listing 9 is an example we will be discussing. We will not be able to discuss everything except for the fundamentals.

```
1 public class com.example.simpleapplication.MainActivity extends
    androidx.appcompat.app.AppCompatActivity
2 {
3     com.example.simpleapplication.Hello contract;
```

```
    private org.web3j.protocol.Web3j web3j;

    public void <init>()
    {
        com.example.simpleapplication.MainActivity $r0;
        com.example.simpleapplication.Hello $r1;

        $r0 := @this: com.example.simpleapplication.MainActivity;

        specialinvoke $r0.<androidx.appcompat.app.
    AppCompatActivity: void <init>()>();

        $r1 = virtualinvoke $r0.<com.example.simpleapplication.
    MainActivity: com.example.simpleapplication.Hello
    ContractInit()>();

        $r0.<com.example.simpleapplication.MainActivity: com.
    example.simpleapplication.Hello contract> = $r1;

        return;
    }

    public com.example.simpleapplication.Hello ContractInit()
    {
        com.example.simpleapplication.MainActivity $r0;
        org.web3j.protocol.http.HttpService $r1;
        org.web3j.protocol.Web3j $r2;
        com.example.simpleapplication.MainActivity$1 $r3;
        org.web3j.crypto.Credentials $r4;
        java.lang.String $r5;
        com.example.simpleapplication.Hello $r6;

        $r0 := @this: com.example.simpleapplication.MainActivity;

        $r1 = new org.web3j.protocol.http.HttpService;

        specialinvoke $r1.<org.web3j.protocol.http.HttpService:
    void <init>(java.lang.String)>("http://10.0.2.2:8545");

        $r2 = staticinvoke <org.web3j.protocol.Web3j: org.web3j.
    protocol.Web3j build(org.web3j.protocol.Web3jService)>($r1);

        $r0.<com.example.simpleapplication.MainActivity: org.
    web3j.protocol.Web3j web3j> = $r2;

```

```
42          $r3 = new com.example.simpleapplication.MainActivity$1;
43
44          specialinvoke $r3.<com.example.simpleapplication.
    MainActivity$1: void <init>(com.example.simpleapplication.
    MainActivity)>($r0);
45
46          $r4 = staticinvoke <org.web3j.crypto.Credentials: org.
    web3j.crypto.Credentials create(java.lang.String)>("0
    x2d2c66151300672e8ec9b8c7315ef744115ff7b82f1e0b9a8bbf39452a3add28
    ");
47
48          $r5 = virtualinvoke $r4.<org.web3j.crypto.Credentials:
    java.lang.String getAddress()>();
49
50          staticinvoke <android.util.Log: int d(java.lang.String,
    java.lang.String)>("Address", $r5);
51
52          $r2 = $r0.<com.example.simpleapplication.MainActivity:
    org.web3j.protocol.Web3j web3j>;
53
54          $r6 = staticinvoke <com.example.simpleapplication.Hello:
    com.example.simpleapplication.Hello load(java.lang.String,org
    .web3j.protocol.Web3j,org.web3j.crypto.Credentials,org.web3j.
    tx.gas.ContractGasProvider)>("0
    x6B0c9E90a645e037949876dCf23523f7185A12bd", $r2, $r4, $r3);
55
56          return $r6;
57      }
58
59      protected void onCreate(android.os.Bundle)
60      {
61          com.example.simpleapplication.MainActivity $r0;
62          android.os.Bundle $r1;
63          org.web3j.protocol.core.RemoteFunctionCall $r2;
64          com.example.simpleapplication.Hello $r3;
65
66          $r0 := @this: com.example.simpleapplication.MainActivity;
67
68          $r1 := @parameter0: android.os.Bundle;
69
70          specialinvoke $r0.<androidx.appcompat.app.
    AppCompatActivity: void onCreate(android.os.Bundle)>($r1);
71
72          virtualinvoke $r0.<com.example.simpleapplication.
    MainActivity: void setContentView(int)>(2131361820);
```

```
73
74          $r3 = $r0.<com.example.simpleapplication.MainActivity:
    com.example.simpleapplication.Hello contract>;
75
76          $r2 = virtualinvoke $r3.<com.example.simpleapplication.
    Hello: org.web3j.protocol.core.RemoteFunctionCall SetName(
    java.lang.String)>("John");
77
78          virtualinvoke $r2.<org.web3j.protocol.core.
    RemoteFunctionCall: java.util.concurrent.CompletableFuture
    sendAsync()>();
79
80          return;
81      }
82 }
```

Listing 9: Jimple Code Example

Listing 9 is an example that we will recreate using the soot Framework. What this code is representing is that we have our Hello smart contract where we are calling the function SetName. The goal is to re-create what will appear in Listing 10.

```
1  package com.example.simpleapplication;
2
3  import android.os.Bundle;
4  import android.util.Log;
5
6  import androidx.appcompat.app.AppCompatActivity;
7
8  import org.web3j.crypto.Credentials;
9  import org.web3j.protocol.Web3j;
10 import org.web3j.protocol.http.HttpService;
11 import org.web3j.tx.gas.ContractGasProvider;
12
13 import java.math.BigInteger;
14
15 public class MainActivity extends AppCompatActivity {
16     private Web3j web3j;
17     Hello contract = ContractInit();
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_main);
23   contract.SetName("John").sendAsync();
```

```
24      }
25      public Hello ContractInit(){
26          web3j = Web3j.build(new HttpService("http
        ://10.0.2.2:8545"));
27
28          ContractGasProvider contractGasProvider = new
        ContractGasProvider() {
29              @Override
30              public BigInteger getGasPrice(String contractFunc) {
31                  return BigInteger.valueOf(3000000);
32              }
33              @Override
34              public BigInteger getGasPrice() {
35                  return BigInteger.valueOf(3000000);
36              }
37              @Override
38              public BigInteger getGasLimit(String contractFunc) {
39                  return BigInteger.valueOf(3000000);
40              }
41              @Override
42              public BigInteger getGasLimit() {
43                  return BigInteger.valueOf(3000000);
44              }
45          };
46
47
48          Credentials credentials = Credentials.create("0
        x2d2c66151300672e8ec9b8c7315ef744115ff7b82f1e0b9a8bbf39452a3add28
        ");
49          Log.d("Address",credentials.getAddress());
50          Hello contract = Hello.load("0
        x6B0c9E90a645e037949876dCf23523f7185A12bd",web3j, credentials
        , contractGasProvider);
51          return contract;
52      }
53  }
```

Listing 10: Android Studio Java Code

What we are trying to do is inject *contract.SetName("John").sendAsync();* at line 23 of our MainActivity.java file located in our Android Application code. Before doing so, we must understand the structure of Android Applications and how they are represented in jimple form.

All the methods presented in the jimple code in Listing 9 are onCreate and ContractInit. The units are a little bit trickier to understand. So let us look at the

method OnCreate represented below in Listing 11. What we should realize is that lines 22 to 54 are the units.

```
1    protected void onCreate(android.os.Bundle)
2    {
3        com.example.simpleapplication.MainActivity $r0;
4        android.os.Bundle $r1;
5        org.web3j.protocol.core.RemoteFunctionCall $r2;
6        com.example.simpleapplication.Hello $r3;
7
8        $r0 := @this: com.example.simpleapplication.MainActivity;
9
10       $r1 := @parameter0: android.os.Bundle;
11
12       specialinvoke $r0.<androidx.appcompat.app.
     AppCompatActivity: void onCreate(android.os.Bundle)>($r1);
13
14       virtualinvoke $r0.<com.example.simpleapplication.
     MainActivity: void setContentView(int)>(2131361820);
15
16       $r3 = $r0.<com.example.simpleapplication.MainActivity:
     com.example.simpleapplication.Hello contract>;
17
18       $r2 = virtualinvoke $r3.<com.example.simpleapplication.
     Hello: org.web3j.protocol.core.RemoteFunctionCall SetName(
     java.lang.String)>("John");
19
20       virtualinvoke $r2.<org.web3j.protocol.core.
     RemoteFunctionCall: java.util.concurrent.CompletableFuture
     sendAsync()>();
21
22       return;
23   }
```

Listing 11: Jimple Code Example Method onCreate

Every unit will contain a statement. For example the statement *$r3 = $r0.<com.example.simpleapplication.MainActivity:com.example.simpleapplication.Hello contract>* consists of a assign statement. There is also a virtualinvoke statement. This means that we are attempting to assign to $r3 the assignment statement for initializing our smart contract call.

We also have references. The references purpose is to tell the compiler where to look for the code that the statements reference. For example if we refer back to our previous statement *$r3=$r0.<com.example.simpleapplication.MainActivity:com.example.simpleapplication.Hello*

*contract*> the $r0 references com.exmaple.simpleapplication.MainActivity on line 8 in Listing 9. An intersting thing about jimple code is that when you are referencing another Java file the statement will include $r[x] = $r[x]<MainFile:SomeOtherJavaFile methodname>. Where *$r[x]* is a reference to the other file location, *MainFile* is a reference to the current Java file, *SomeOther-JavaFile* is a reference to the other Java file containing the method you are trying to inject, and *methodname* is the method you want to call.

Based on the previous details, notice that in jimple code, all references are inside each method and are represented by some text followed with a $r[x] where [x] represents some integer value followed by a semicolon. This is important to understand, especially a vital detail to keep in mind for the next section, "Injecting Blockchain Calls into Android Applications."

## 7.3 Injecting Blockchain Calls into Android Applications

In this section we want to introduce how to inject blockchain calls using the Soot framework. The blockchain call we will be injecting is utilizing the Android Studio project we set-up earlier.

### Generating the APK to analyze

We want to create a APK file to analyse.
This is accomplished by:

1. opening up our Android Studio project and clicking on $build \rightarrow BuildBundle(s)/APK(s) \rightarrow BuildAPK(s)$.

2. Once you see the message shown in Figure 19 Click on the locate button.

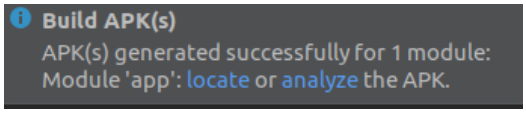


Figure 19: Build APK Message

3. Rename the file to *SimpleApplication.apk* and paste it into a new directory on your file system.

   Be sure to remember where this file was saved to and write it down. This file location will be used later on by the Soot Framework.

### 7.3.1 Setting up the Eclipse Workspace and including Soot

Eclipse is a popular integrated development environment that easily enables developers to generate and work with Java code. You will want to install Eclipse by going to `https://www.eclipse.org/downloads/` or installing directly from the Ubuntu software store.

Now we need to set-up our new project. This is accomplished by:

1. Start Eclipse
2. Next, create a new Java project by clicking on *File* $\rightarrow$ *New Java Project*.

   Then you will see a blank Java Project with an empty class Java File.
3. When the set up wizard appears, click on $Java \rightarrow JavaProject$ and then click Next.
4. Fill in the information as displayed in Figure 20. Note, even though I have an error message, I have presented you with a project with the same name already existing in the current path of my Linux file system. You should be able to click next without any errors.

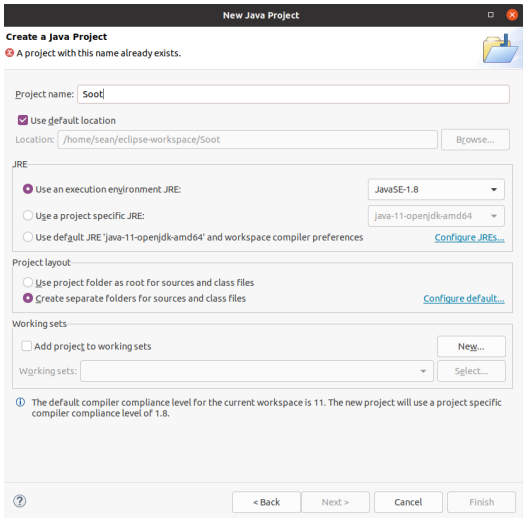


Figure 20: Eclipse Java Project Set-up

5. For the next screen, be sure to accept the defaults, and you should now have an empty project created.

6. Create a new class file by clicking on the drop-down menu for $Soot \rightarrow src$ and then right-clicking on src and selecting $new \rightarrow class$.

7. When the menu appears, type in the Name field *HicssSootAnalysis*.

   This will create for you an empty Java class file.

8. Next, we will want to copy and paste the contents from Listing 12 into the Java class file we just created.

   Do not worry about what the code means yet. We will explain it in detail later.

```
1  import soot.Body;
2  import soot.BodyTransformer;
3  import soot.Local;
4  import soot.PackManager;
5  import soot.PatchingChain;
6  import soot.RefType;
7  import soot.Scene;
8  import soot.SootClass;
9  import soot.SootMethodRef;
10 import soot.SootResolver;
11 import soot.Transform;
12 import soot.Type;
13 import soot.Unit;
14 import soot.Value;
15 import soot.VoidType;
16
17
18 import java.util.*;
19
20 public class HicssSootAnalysis
21 {
22
23     static String EntryPointApp = "com.example.
       simpleapplication.MainActivity";
24     static String EntryPointBlockchain = "com.example.
       simpleapplication.Hello";
25     static String RemoteFunctionCallWeb3J = "org.web3j.
       protocol.core.RemoteFunctionCall";
26     static String SendAsync = "org.web3j.protocol.core.
       RemoteFunctionCall: java.util.concurrent.
       CompletableFuture sendAsync()";
27     static Local ContractReference = null;
28     static Value BlockchainContractStmt = null;
```

```
29
30    public static void main(String[] args)
31    {
32
33         PackManager.v().getPack("jtp").add(new Transform("
      jtp.myInstrumenter", new BodyTransformer() {
34            @Override
35            protected void internalTransform(final Body body
      , String phaseName, @SuppressWarnings("rawtypes") Map
      options) {
36                final PatchingChain<Unit> units = body.
      getUnits();
37                IterateOverAllUnits(units, body);
38            }
39        }));
40        soot.Main.main(args);
41    }
42
43    private static void IterateOverAllUnits(PatchingChain<
      Unit> units, Body body) {
44        String StringLastKnownUnit = "";
45        int counter = 0;
46        for(Iterator<Unit> UnitIterator = units.
      snapshotIterator(); UnitIterator.hasNext();) {
47            final Unit LastKnownUnit = UnitIterator.next();
48            StringLastKnownUnit = LastKnownUnit.toString();
49        }
50
51    }
52    public static SootMethodRef makeMethodRef(String cName,
      String mName, String rType, List<String> pTypes,
53            boolean isStatic) {
54           SootClass sc = SootResolver.v().makeClassRef(
      cName);
55           Type returnType = null;
56           if (rType == "") {
57            returnType = VoidType.v();
58           } else {
59            returnType = RefType.v(rType);
60           }
61           List<Type> parameterTypes = new ArrayList<Type>()
      ;
62           for (String p : pTypes) {
63            parameterTypes.add(RefType.v(p));
64           }
```

```
65              return Scene.v().makeMethodRef(sc, mName,
    parameterTypes, returnType, isStatic);
66      }
67
68      private static Local Return_Local(Body b, String name) {
69          for (Local local : b.getLocals()) {
70             if (local.getType().toString().contains(name)) {
71              return local;
72             }
73          }
74          return null;
75      }
76
77
78      private static boolean Check_If_SetContentView_Exists(
    String StringLastKnownUnit) {
79          if(Contains_Match_Case(StringLastKnownUnit,
    EntryPointApp) && Contains_Match_Case(
    StringLastKnownUnit, "SetContentView")) {
80              return true;
81          }else {
82              return false;
83          }
84      }
85
86
87      public static boolean Contains_Match_Case(String str,
    String subString) {
88          return str.toLowerCase().contains(subString.
    toLowerCase());
89      }
90  }
```

Listing 12: Eclipse Java Code

## 7.4 Including Soot in the Eclipse Project

Now we need to include the Soot Framework in our eclipse project. Begin by:

1. create a Maven project by following the instructions at
   `https://crunchify.com/how-to-convert-existing-java-project-to-maven-in-eclipse/`.
2. Navigate to our *pom.xml* file and copy the code from listing 13 to our file. Then save the file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi
   ="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>Soot</groupId>
  <artifactId>Soot</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
  <dependency>
    <groupId>ca.mcgill.sable</groupId>
    <artifactId>soot</artifactId>
    <version>4.0.0</version>
</dependency>
  </dependencies>
</project>
```

Listing 13: XML file

3.

### 7.4.1 Injecting our Blockchain Call using Eclipse

This section will explain how to inject the blockchain call into the Android Application.
For injecting the blockchain call we will want to add the following lines as shown in Listing 14 to our *HicssSootAnalysis.java* file in Eclipse at line 49.

```
if(Check_If_SetContentView_Exists(StringLastKnownUnit)) {
                //ADD LOCALS
```

```
3               LocalGenerator l = new LocalGenerator(body);
4               Local lgContractReference = l.generateLocal(
    RefType.v("com.example.simpleapplication.Hello"));
5               Local lgRemoteFunctionCallReference = l.
    generateLocal(RefType.v("org.web3j.protocol.core.
    RemoteFunctionCall"));
6
7               // Create $r3 = $r0.<com.example.
    simpleapplication.MainActivity: com.example.simpleapplication
    .Hello contract>;
8               LinkedList<Value> ContractInitArgs = new
    LinkedList<>();
9               AssignStmt AssignStmtContract = Jimple.v().
    newAssignStmt(Return_Local(body, "Hello"),Jimple.v().
    newVirtualInvokeExpr(Return_Local(body,"MainActivity"), Scene
    .v().getMethod("<"+ EntryPointApp +": "+ EntryPointBlockchain
     +" ContractInit()>").makeRef(), ContractInitArgs));
10              units.insertAfter(AssignStmtContract,
    LastKnownUnit);
11              // Create $r2 = virtualinvoke $r3.<com.example.
    simpleapplication.Hello: org.web3j.protocol.core.
    RemoteFunctionCall SetName(java.lang.String)>("John");
12              LinkedList<Value> ContractPersonName = new
    LinkedList<>();
13              ContractPersonName.add(StringConstant.v("John"));
14              AssignStmt AssignStmtContractFunc = Jimple.v().
    newAssignStmt(Return_Local(body, "RemoteFunctionCall"),Jimple
    .v().newVirtualInvokeExpr(Return_Local(body,"Hello"), Scene.v
    ().getMethod("<"+ EntryPointBlockchain +": "+
    RemoteFunctionCallWeb3J +" SetName(java.lang.String)>").
    makeRef(), ContractPersonName));
15              units.insertAfter(AssignStmtContractFunc,
    AssignStmtContract);
16              // Create virtualinvoke $r2.<org.web3j.protocol.
    core.RemoteFunctionCall: java.util.concurrent.
    CompletableFuture sendAsync()>();
17              List<String> BlockchainArgumentsForSendAsync =
    new LinkedList<>();
18              SootMethodRef SendAsyncMethodRef = makeMethodRef(
    "org.web3j.protocol.core.RemoteCall","sendAsync", "java.util.
    concurrent.CompletableFuture",BlockchainArgumentsForSendAsync
    , false);
19              VirtualInvokeExpr InvokeExprSendAsync = Jimple.v
    ().newVirtualInvokeExpr(Return_Local(body,"RemoteFunctionCall
    "), SendAsyncMethodRef, Collections.<Value>emptyList());
```

```
                InvokeStmt InvokeStatementSendAsync = Jimple.v().
    newInvokeStmt(InvokeExprSendAsync);
                units.insertAfter(InvokeStatementSendAsync, (Unit
    ) AssignStmtContractFunc);
}
```

Listing 14: Code to Inject

After the code will look similar to Listing 15.

```
import soot.Body;
import soot.BodyTransformer;
import soot.Local;
import soot.PackManager;
import soot.PatchingChain;
import soot.RefType;
import soot.Scene;
import soot.SootClass;
import soot.SootMethodRef;
import soot.SootResolver;
import soot.Transform;
import soot.Type;
import soot.Unit;
import soot.Value;
import soot.VoidType;
import soot.javaToJimple.LocalGenerator;
import soot.jimple.AssignStmt;
import soot.jimple.InvokeStmt;
import soot.jimple.Jimple;
import soot.jimple.StringConstant;
import soot.jimple.VirtualInvokeExpr;

import java.util.*;

public class HicssSootAnalysis
{

    static String EntryPointApp = "com.example.simpleapplication.
    MainActivity";
    static String EntryPointBlockchain = "com.example.
    simpleapplication.Hello";
    static String RemoteFunctionCallWeb3J = "org.web3j.protocol.
    core.RemoteFunctionCall";
    static String SendAsync = "org.web3j.protocol.core.
    RemoteFunctionCall: java.util.concurrent.CompletableFuture
    sendAsync()";
    static Local ContractReference = null;
```

```
33      static Value BlockchainContractStmt = null;
34
35      public static void main(String[] args)
36      {
37
38          PackManager.v().getPack("jtp").add(new Transform("jtp.
    myInstrumenter", new BodyTransformer() {
39              @Override
40              protected void internalTransform(final Body body,
    String phaseName, @SuppressWarnings("rawtypes") Map options)
    {
41                  final PatchingChain<Unit> units = body.getUnits()
    ;
42                  IterateOverAllUnits(units, body);
43              }
44          }));
45          soot.Main.main(args);
46      }
47
48      private static void IterateOverAllUnits(PatchingChain<Unit>
    units, Body body) {
49          String StringLastKnownUnit = "";
50          int counter = 0;
51          for(Iterator<Unit> UnitIterator = units.snapshotIterator
    (); UnitIterator.hasNext();) {
52              final Unit LastKnownUnit = UnitIterator.next();
53              StringLastKnownUnit = LastKnownUnit.toString();
54              if(Check_If_SetContentView_Exists(StringLastKnownUnit
    )) {
55                //ADD LOCALS
56                  LocalGenerator l = new LocalGenerator(body);
57                  Local lgContractReference = l.generateLocal(
    RefType.v("com.example.simpleapplication.Hello"));
58                  Local lgRemoteFunctionCallReference = l.
    generateLocal(RefType.v("org.web3j.protocol.core.
    RemoteFunctionCall"));
59
60                  // Create $r3 = $r0.<com.example.
    simpleapplication.MainActivity: com.example.simpleapplication
    .Hello contract>;
61                  LinkedList<Value> ContractInitArgs = new
    LinkedList<>();
62                  AssignStmt AssignStmtContract = Jimple.v().
    newAssignStmt(Return_Local(body, "Hello"),Jimple.v().
    newVirtualInvokeExpr(Return_Local(body,"MainActivity"), Scene
```

```
.v().getMethod("<"+ EntryPointApp +": "+ EntryPointBlockchain
 +" ContractInit()>").makeRef(), ContractInitArgs));
63              units.insertAfter(AssignStmtContract,
LastKnownUnit);
64              // Create $r2 = virtualinvoke $r3.<com.example.
simpleapplication.Hello: org.web3j.protocol.core.
RemoteFunctionCall SetName(java.lang.String)>("John");
65              LinkedList<Value> ContractPersonName = new
LinkedList<>();
66              ContractPersonName.add(StringConstant.v("John"));
67              AssignStmt AssignStmtContractFunc = Jimple.v().
newAssignStmt(Return_Local(body, "RemoteFunctionCall"),Jimple
.v().newVirtualInvokeExpr(Return_Local(body,"Hello"), Scene.v
().getMethod("<"+ EntryPointBlockchain +": "+
RemoteFunctionCallWeb3J +" SetName(java.lang.String)>").
makeRef(), ContractPersonName));
68              units.insertAfter(AssignStmtContractFunc,
AssignStmtContract);
69              // Create virtualinvoke $r2.<org.web3j.protocol.
core.RemoteFunctionCall: java.util.concurrent.
CompletableFuture sendAsync()>();
70              List<String> BlockchainArgumentsForSendAsync =
new LinkedList<>();
71              SootMethodRef SendAsyncMethodRef = makeMethodRef(
"org.web3j.protocol.core.RemoteCall","sendAsync", "java.util.
concurrent.CompletableFuture",BlockchainArgumentsForSendAsync
, false);
72              VirtualInvokeExpr InvokeExprSendAsync = Jimple.v
().newVirtualInvokeExpr(Return_Local(body,"RemoteFunctionCall
"), SendAsyncMethodRef, Collections.<Value>emptyList());
73              InvokeStmt InvokeStatementSendAsync = Jimple.v().
newInvokeStmt(InvokeExprSendAsync);
74              units.insertAfter(InvokeStatementSendAsync, (Unit
) AssignStmtContractFunc);
75          }
76      }
77
78  }
79  public static SootMethodRef makeMethodRef(String cName,
String mName, String rType, List<String> pTypes,
80          boolean isStatic) {
81         SootClass sc = SootResolver.v().makeClassRef(cName);
82         Type returnType = null;
83         if (rType == "") {
84          returnType = VoidType.v();
```

```
            } else {
             returnType = RefType.v(rType);
            }
            List<Type> parameterTypes = new ArrayList<Type>();
            for (String p : pTypes) {
             parameterTypes.add(RefType.v(p));
            }
            return Scene.v().makeMethodRef(sc, mName,
    parameterTypes, returnType, isStatic);
     }

     private static Local Return_Local(Body b, String name) {
         for (Local local : b.getLocals()) {
            if (local.getType().toString().contains(name)) {
             return local;
            }
         }
         return null;
    }


     private static boolean Check_If_SetContentView_Exists(String
    StringLastKnownUnit) {
         if(Contains_Match_Case(StringLastKnownUnit, EntryPointApp
    ) && Contains_Match_Case(StringLastKnownUnit, "SetContentView
    ")) {
             return true;
         }else {
             return false;
         }
     }


     public static boolean Contains_Match_Case(String str, String
    subString) {
         return str.toLowerCase().contains(subString.toLowerCase()
    );
     }
}
```

Listing 15: Soot Analysis Code With Blockchain Injection Code

Now lets look at the following:

```
            if(Check_If_SetContentView_Exists(StringLastKnownUnit
```

```
    )) {
```

Listing 16: HicssSootAnalysis Code Snippet

What is represented here is a simple function in a if condition that will look and check if the SetContentView has been found. We know that based off of our previous Android Code that SetContentView is in our MainActivity. This eliminates Soot ;having to know where the entry point is at.

```
1                 LocalGenerator l = new LocalGenerator(body);
2                 Local lgContractReference = l.generateLocal(
    RefType.v("com.example.simpleapplication.Hello"));
3                 Local lgRemoteFunctionCallReference = l.
    generateLocal(RefType.v("org.web3j.protocol.core.
    RemoteFunctionCall"));
```

Listing 17: HicssSootAnalysis Code Snippet

The code above is generating our locals. This is referencing our remote function call and the Hello smart contract Java code. The following lines are required and without them the injection would not work.

```
1                 LinkedList<Value> ContractInitArgs = new
    LinkedList<>();
2                 AssignStmt AssignStmtContract = Jimple.v().
    newAssignStmt(Return_Local(body, "Hello"),Jimple.v().
    newVirtualInvokeExpr(Return_Local(body,"MainActivity"), Scene
    .v().getMethod("<"+ EntryPointApp +": "+ EntryPointBlockchain
     +" ContractInit()>").makeRef(), ContractInitArgs));
3                 units.insertAfter(AssignStmtContract,
    LastKnownUnit);
```

Listing 18: HicssSootAnalysis Code Snippet

The code above is used to generate the code
*$r3 = $r0.<com.example.simpleapplication.MainActivity: com.example.simpleapplication.Hello contract>*. On line 2 we have to generate a linked list that will take in no parameters. Then on Line 3 we create a new assign statement with a virtualinvoke expression inside. Then we insert after the last unit that was discovered by Soot.

```
1                 LinkedList<Value> ContractPersonName = new
    LinkedList<>();
2                 ContractPersonName.add(StringConstant.v("John"));
3                 AssignStmt AssignStmtContractFunc = Jimple.v().
    newAssignStmt(Return_Local(body, "RemoteFunctionCall"),Jimple
    .v().newVirtualInvokeExpr(Return_Local(body,"Hello"), Scene.v
```

```
    ().getMethod("<"+ EntryPointBlockchain +": "+
    RemoteFunctionCallWeb3J +" SetName(java.lang.String)>").
    makeRef(), ContractPersonName));
4               units.insertAfter(AssignStmtContractFunc,
    AssignStmtContract);
```

Listing 19: Code Snippet

What we wanted to do is pass the persons name to our smart contract. We accomplished this by passing in John into another linked list as displayed in the code above. Also we wanted to create another assignment statement with another virtualinvoke expression inside. This resulted in the following jimple code:
*$r2 = virtualinvoke $r3.<com.example.simpleapplication.Hello: org.web3j.protocol.core.RemoteFunctionCall SetName(java.lang.String)>("John").*

```
1               List<String> BlockchainArgumentsForSendAsync =
    new LinkedList<>();
2               SootMethodRef SendAsyncMethodRef = makeMethodRef(
    "org.web3j.protocol.core.RemoteCall","sendAsync", "java.util.
    concurrent.CompletableFuture",BlockchainArgumentsForSendAsync
    , false);
3               VirtualInvokeExpr InvokeExprSendAsync = Jimple.v
    ().newVirtualInvokeExpr(Return_Local(body,"RemoteFunctionCall
    "), SendAsyncMethodRef, Collections.<Value>emptyList());
4               InvokeStmt InvokeStatementSendAsync = Jimple.v().
    newInvokeStmt(InvokeExprSendAsync);
5               units.insertAfter(InvokeStatementSendAsync, (Unit
    ) AssignStmtContractFunc);
```

Listing 20: Code Snippet

We now needed to create the following jimple code:
*virtualinvoke $r2.<org.web3j.protocol.core.RemoteFunctionCall: java.util.concurrent.CompletableFuture sendAsync()>()*. This was accomplished via creating a empty linked list as displayed on line 1 and then creatinging another Soot method reference as displayed on line 2. Then we had to create a virtualinvoke expression as displayed on line 3. Finally we had to create a invoke statement and insert it after the last known unit that Soot found. Note, that the last unit you insert for SendAsync is much different that the first 2 code snippets we discussed here.

### 7.4.2 Running Eclipse Example

To generate the APK with the code that was just created we need to set-up the Eclipse arguments to be able to generate the APK. To set the arguments:

1. Click on the drop-down menu button  , select **Run Configurations...**, and then click on the Arguments tab.

2. Copy and paste the arguments displayed in Listing 21.

   Note, *Android Jar Location* is the location where your Android Jar is located in your filesystem it is usually */home/username/Android/Sdk/platforms* where username is replaced with your username. Also, *APK to Analyse* is the location where you stored the APK that you want to inject the blockchain calls into.

```
-allow-phantom-refs -android-jars [Android Jar Location] -
   android-api-version 28 -src-prec apk -output-format dex
   -force-overwrite -output-dir OutputAPK -process-dir [APK
    to Analyse Location] --process-multiple-dex -p db.
   transformations enabled:true
```

Listing 21: Arguments

## 7.5 Signing our generated APK

Now we need to sign our Android APK file so that our phone emulator will be able to run our Android APK file. Without signing it the emulator would reject the APK from ever running. In order to sign the APK file we will need to refer to `https://stackoverflow.com/questions/50705658/how-to-sign-an-apk-through-command-line`.

## 7.6 Installing the APK on the phone emulator

Now we have signed our APK and we want to install it on our phone emulator. To install the APK on our phone emulator:

1. Open up our command prompt window and type in *emulator -list-avds*.

   This command will allow you to see what phone emulators you have installed. You will want to copy the text of the first phone emulator you see. If you do not see anything you will need to go and install a phone emulator as previously mentioned in our Android Set-up section.

2. Type in **emulator -avd** and paste in the previous text we copied.

   This will start the emulator for you.

3. Change directories to the location of where our generated APK is at. You can find this location by going to your Eclipse window and right clicking on Properties and selecting the Location path.

4. Paste the path when issuing the cd command.

5. Type in *adb install SimpleApplication.apk*.

   Once the success message appears, you have now properly installed the APK on your phone emulator.

## 7.7 Running the Private Blockchian and getting everything working

The steps for getting everything to communicate are:

1. Make sure Ganache is running by clicking on the AppImage file.

2. Click on *Quickstart* button.

3. Start the phone emulator.

4. Pass our smart contract to the private blockchain by going to `http://remix.ethereum.org/` and making sure we follow the steps provided in the Back to Remix section.

   It is now time to check to see if some data was passed to Ganache. This is accomplished via going to the *TRANSACTIONS* tab and making sure that we see that there is a contract call. It will be a light blue message that says *CONTRACT CALL* next to the transaction.

## 7.8 Injecting blockchain code into Android applications using APKTool

### 7.8.1 Smali Code

The APKtool is one of the most exciting and useful reverse engineering tools because it can easily decompile an Android APK into smali code. The smali code can then be modified or altered. This modified code can then be converted into an APK. Smali code is essentially the assembly language that runs on Android's Java virtual machine. The modification of smali code takes significant technical expertise. It requires a considerable amount of time to understand the register structure and the fine details of smali code modification.

For example, let us take a look at listing 22.

```
.class public Lcom/example/simpleapplication/MainActivity;
.super Landroidx/appcompat/app/AppCompatActivity;
.source "MainActivity.java"


# instance fields
.field contract:Lcom/example/simpleapplication/Hello;

.field private web3j:Lorg/web3j/protocol/Web3j;


# direct methods
.method public constructor <init>()V
    .locals 1

    .line 15
    invoke-direct {p0}, Landroidx/appcompat/app/AppCompatActivity
    ;-><init>()V

    .line 17
    invoke-virtual {p0}, Lcom/example/simpleapplication/
    MainActivity;->ContractInit()Lcom/example/simpleapplication/
    Hello;

    move-result-object v0

    iput-object v0, p0, Lcom/example/simpleapplication/
    MainActivity;->contract:Lcom/example/simpleapplication/Hello;

```

```
26      return-void
27  .end method
```

Listing 22: Smali File Example

APKTool generates comments for the instance fields, direct methods, and virtual methods. This makes it quicker to analyse see what is going on. The *.method* keyword is used to identify a method. The *.class* keyword is used to tell what class the specific smali code file belongs to.
The *.locals* keywork is used to define how many registers were used in the specific method. The registers will always begin the v[x] where [x] represents an integer value. The *.line* indicates what line in the file the smali code refers to. Note that *.line* could contain multiple lines of smali code. So every new line that is referenced is incremented by some integer value. For example, in the example file line 16 references the .line 15 and line 19 references .line 17 and spans multiple lines.
For more information visit `https://github.com/JesusFreke/smali/wiki/Registers`

### 7.8.2 Injecting Blockchain Calls into Android Applications using APKTool

Let us now discuss how to inject blockchain calls using APKTool.

The instructions are as follows:

1. Decompile our example Android APK file using the following command: **apktool -r d SimpleApplication.apk**.

   This command will decompile our APK and all contents into a folder named SimpleApplication. The folder will contain the following folder structure, as displayed in Figure 21.

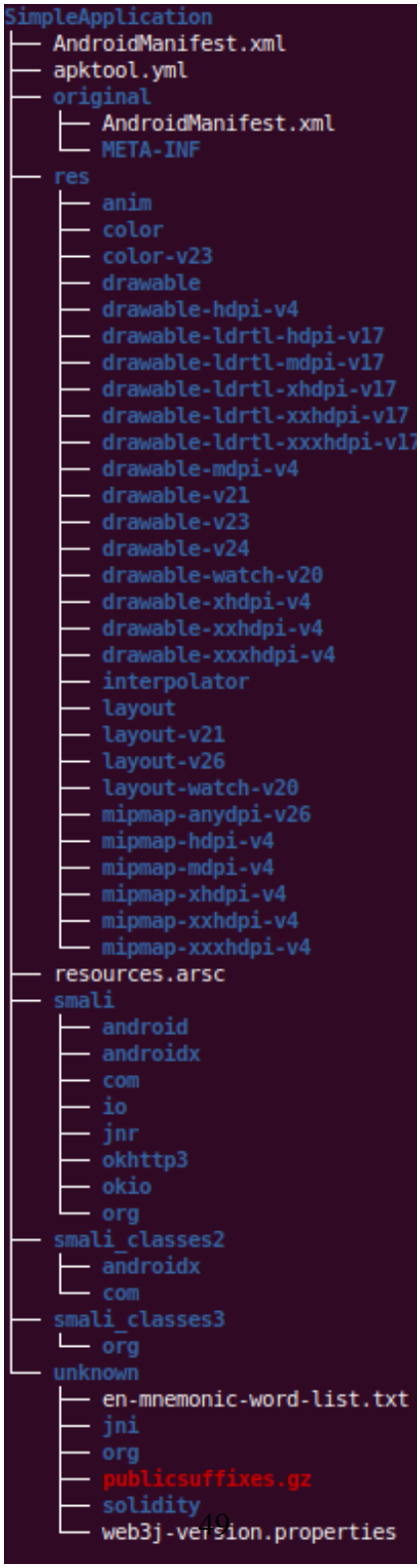


Figure 21: APKTool SimpleApplication Folder Structure

We only need to be concerned with the folder $SimpleApplication/smali_classes2/com/example/simpleapplication$. Also, we will only be manipulating the MainActivity.smali file.

2. What we want to inject into our MainActivity.smali file at line 95 is the following lines that appear in listing 23.

```
    .line 23
    iget-object v0, p0, Lcom/example/simpleapplication/
   MainActivity;->contract:Lcom/example/simpleapplication/
   Hello;

    const-string v1, "John"

    invoke-virtual {v0, v1}, Lcom/example/simpleapplication/
   Hello;->SetName(Ljava/lang/String;)Lorg/web3j/protocol/
   core/RemoteFunctionCall;

    move-result-object v0

    invoke-virtual {v0}, Lorg/web3j/protocol/core/
   RemoteFunctionCall;->sendAsync()Ljava/util/concurrent/
   CompletableFuture;
```

Listing 23: Smali Code to Inject

3. Increment the integer value by one for line 85. The final output should look like listing 24.

```
.class public Lcom/example/simpleapplication/MainActivity;
.super Landroidx/appcompat/app/AppCompatActivity;
.source "MainActivity.java"


# instance fields
.field contract:Lcom/example/simpleapplication/Hello;

.field private web3j:Lorg/web3j/protocol/Web3j;


# direct methods
.method public constructor <init>()V
    .locals 1

    .line 15
```

```
    invoke-direct {p0}, Landroidx/appcompat/app/
    AppCompatActivity;-><init>()V

    .line 17
    invoke-virtual {p0}, Lcom/example/simpleapplication/
    MainActivity;->ContractInit()Lcom/example/
    simpleapplication/Hello;

    move-result-object v0

    iput-object v0, p0, Lcom/example/simpleapplication/
    MainActivity;->contract:Lcom/example/simpleapplication/
    Hello;

    return-void
.end method


# virtual methods
.method public ContractInit()Lcom/example/simpleapplication/
    Hello;
    .locals 4

    .line 26
    new-instance v0, Lorg/web3j/protocol/http/HttpService;

    const-string v1, "http://10.0.2.2:8545"

    invoke-direct {v0, v1}, Lorg/web3j/protocol/http/
    HttpService;-><init>(Ljava/lang/String;)V

    invoke-static {v0}, Lorg/web3j/protocol/Web3j;->build(
    Lorg/web3j/protocol/Web3jService;)Lorg/web3j/protocol/
    Web3j;

    move-result-object v0

    iput-object v0, p0, Lcom/example/simpleapplication/
    MainActivity;->web3j:Lorg/web3j/protocol/Web3j;

    .line 28
    new-instance v0, Lcom/example/simpleapplication/
    MainActivity$1;

    invoke-direct {v0, p0}, Lcom/example/simpleapplication/
```

```
    MainActivity$1;-><init>(Lcom/example/simpleapplication/
    MainActivity;)V
51
52      .line 48
53      .local v0, "contractGasProvider":Lorg/web3j/tx/gas/
    ContractGasProvider;
54      const-string v1, "0
    x2d2c66151300672e8ec9b8c7315ef744115ff7b82f1e0b9a8bbf39452a3add28
    "
55
56      invoke-static {v1}, Lorg/web3j/crypto/Credentials;->
    create(Ljava/lang/String;)Lorg/web3j/crypto/Credentials;
57
58      move-result-object v1
59
60      .line 49
61      .local v1, "credentials":Lorg/web3j/crypto/Credentials;
62      invoke-virtual {v1}, Lorg/web3j/crypto/Credentials;->
    getAddress()Ljava/lang/String;
63
64      move-result-object v2
65
66      const-string v3, "Address"
67
68      invoke-static {v3, v2}, Landroid/util/Log;->d(Ljava/lang
    /String;Ljava/lang/String;)I
69
70      .line 50
71      iget-object v2, p0, Lcom/example/simpleapplication/
    MainActivity;->web3j:Lorg/web3j/protocol/Web3j;
72
73      const-string v3, "0
    x6B0c9E90a645e037949876dCf23523f7185A12bd"
74
75      invoke-static {v3, v2, v1, v0}, Lcom/example/
    simpleapplication/Hello;->load(Ljava/lang/String;Lorg/
    web3j/protocol/Web3j;Lorg/web3j/crypto/Credentials;Lorg/
    web3j/tx/gas/ContractGasProvider;)Lcom/example/
    simpleapplication/Hello;
76
77      move-result-object v2
78
79      .line 51
80      .local v2, "contract":Lcom/example/simpleapplication/
    Hello;
```

```
    return-object v2
.end method

.method protected onCreate(Landroid/os/Bundle;)V
    .locals 2
    .param p1, "savedInstanceState"    # Landroid/os/Bundle;

    .line 21
    invoke-super {p0, p1}, Landroidx/appcompat/app/
    AppCompatActivity;->onCreate(Landroid/os/Bundle;)V

    .line 22
    const v0, 0x7f0a001c

    invoke-virtual {p0, v0}, Lcom/example/simpleapplication/
    MainActivity;->setContentView(I)V

    .line 23
    iget-object v0, p0, Lcom/example/simpleapplication/
    MainActivity;->contract:Lcom/example/simpleapplication/
    Hello;

    const-string v1, "John"

    invoke-virtual {v0, v1}, Lcom/example/simpleapplication/
    Hello;->SetName(Ljava/lang/String;)Lorg/web3j/protocol/
    core/RemoteFunctionCall;

    move-result-object v0

    invoke-virtual {v0}, Lorg/web3j/protocol/core/
    RemoteFunctionCall;->sendAsync()Ljava/util/concurrent/
    CompletableFuture;

    .line 24
    return-void
.end method
```

Listing 24: Smali Code Final Result

4. Compile our code by issuing the following command:

```
apktool b simpleAd
```

### 7.8.3 Installing the Generated APK file

It is now time to install the generated APK file.

Do so by:

1. Navigating to the folder $SimpleApplication/dist/$.
2. Sign the APK as described in the section *Signing our generated APK*.
3. Install the APK using the command: *adb install SimpleApplication.apk*.

   Note, it is a requirement to uninstall the previous application with the same application name. If not, you will be unable to install the correct APK on your phone emulator.

# References


[1] L. Mearian, "What is blockchain? The complete guide," Jan. 2019, library Catalog: www.computerworld.com. [Online]. Available: https://www.computerworld.com/article/3191077/what-is-blockchain-the-complete-guide.html

[2] "Ganache | Ganache Quickstart | Documentation," library Catalog: www.trufflesuite.com. [Online]. Available: https://trufflesuite.com/docs/ganache/quickstart

[3] "soot-oss/soot," library Catalog: github.com. [Online]. Available: https://github.com/soot-oss/soot

[4] "Apktool - Documentation." [Online]. Available: https://ibotpeaches.github.io/Apktool/documentation/