White Paper

McAfee®
An Intel Company

Defeat SSL Certificate Validation for Google Android Applications

By Naveen Rudrappa, Security Consultant,
McAfee® Foundstone® Professional Services

Table of Contents

Overview

Proxy tools like Fiddler, Paros, Web Scarab, and Burp Suite have been great assets for penetration testers while performing web application and thick client security testing. These tools allow penetration testers to intercept and forward the HTTP(S) traffic to and from the client application. Proxy tools are easy to use for such applications because testers can directly add their web proxy certificate to the trusted certification authority (CA) certificate store on the test machine.

However, adding a certificate to trusted CA store is not trivial when working with mobile applications. Workarounds have been documented for a number of mobile platforms to accept any CA as a trusted, except for Google Android OS.

This paper discusses a workaround to skip SSL certificate verification so that you can route HTTPS traffic for Android-based mobile applications through any proxy tool. The workaround involves injecting extra classes and modifying the existing code so that the application skips the SSL certificate verification task.

This white paper is intended to assist penetration testers working on Android-based applications.

Proxy Primer

Proxy tools are based on the man-in-the-middle (MITM) concept. Proxy tools are inserted between the server and the client. After being inserted, the proxy intercepts the traffic and relays messages between them. During this process, if SSL is in use, proxy tools can decrypt the traffic as well. They do this by setting up two SSL connections: one between itself and the web server and another between itself and the client.

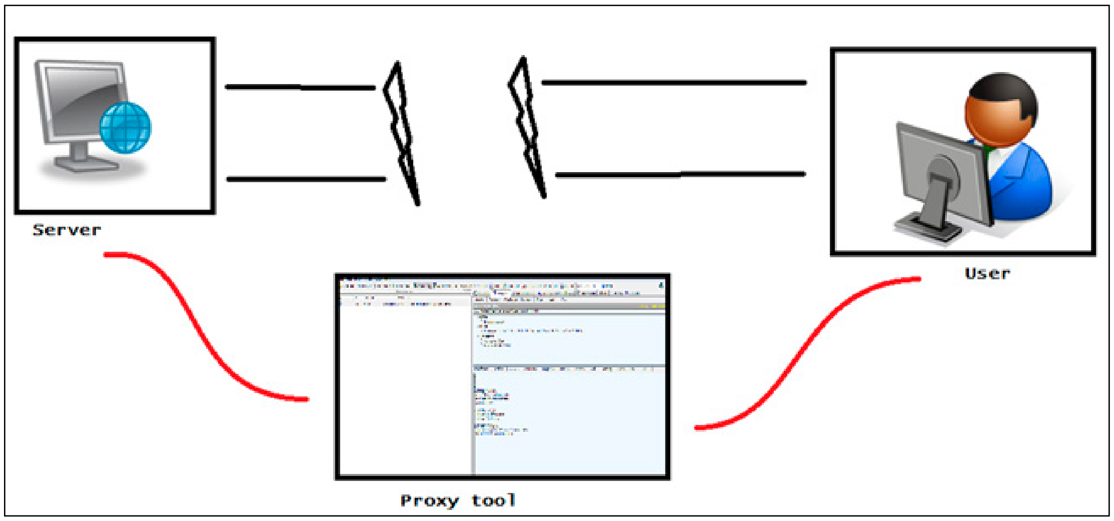


Figure 1. How proxy tools work.

When setting up the SSL connection with the client, the proxy tool provides its own certificate in lieu of the server certificate. This certificate is then used during the SSL handshake to establish the SSL connection between the proxy tool and the client. When the client receives the certificate, the code responsible for setting up the SSL connection in the client checks to see whether the certificate is signed by a trusted CA by querying the certificate manager. Since the proxy certificate root is not among the trusted CAs, an SSL connection is not established unless the user overrides this decision. Therefore, to establish a successful SSL connection with the proxy tool, the CA certificate of the proxy tool has to be pushed into the trusted zone of the certificate manager, also known as keystore on the client.

Validation of SSL Certificates in Android OS

Every OS has its own keystore, which contains all the trusted CAs. If a user wants to add a proxy tool CA into the trusted zone, he has to add the proxy tool's signing certificate into the trusted zone of the certificate manager. This is difficult on Android OS since it does not allow non-trusted CA certificates to be added into the keystore.

Android OS has the trusted CA root certificate stored in a keystore at the following location:

```
/system/etc/security/cacerts.bks
```

Android does not allow addition of custom certificates to its keystore. We could try to forcibly insert a certificate into the Android keystore. Android has to be restarted to cause it to accept the inserted certificate as a trusted one. Upon restarting, however, the Android OS deletes all the user-added certificates. This behavior of the Android OS is a major challenge to routing HTTPS traffic through a proxy tool because there is no way for the proxy tool's root certificate to be added to the trusted zone.

Routing HTTPS Traffic via a Proxy

Given the discussion above, our best way forward appears to be to skip the SSL certificate check in the Android application that is being installed. To do so, we will have to overwrite the code that performs the SSL certificate verification so that no exception is thrown while accepting a certificate signed by a non-trusted CA.

Before modifying the decompiled code of an Android installable, let's understand how we can go about overcoming SSL certificate validation if the source code is available.

There are three main classes in Java that are used to access web applications. These classes are listed below:

- `android.webkit.WebView`
- `javax.net.ssl.HttpsURLConnection`
- `org.apache.http.impl.client.DefaultHttpClient`

Overcoming SSL certificate validation for each of these classes involves injecting different code.

Code Injection

Like other object-oriented languages, Java overrides methods that allow a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. The implementation in the subclass overrides (replaces) the implementation in the super-class by providing a method that has the same name, same parameters, and the same return type as the method in the parent class—in other words, an identical signature.

This concept is used to override the implementation of functions in parent class to skip SSL certificate verification during initial SSL handshake.

Defeating SSL Certificate Validation When Application Source Is Available

org.apache.http.impl.client.DefaultHttpClient

Step 1:

Create the classes listed below:

- EasyX509TrustManager.java
- EasySSLSocketFactory.java
- MyHttpClient.java

Refer to Section 1 of the appendix for the code.

Step 2:
Add these classes to the existing source code in package `com.android`.

Step 3:
Replace all the instances of `DefaultHttpClient` with `MyHttpClient`.

Example: `DefaultHttpClient client = new MyHttpClient();`

Step 4:
Compile the project to generate the installable.

Step 5:
Digitally sign the application using the following commands:

- `keytool –genk ey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000`
- `jarsigner -verbose -keystore my-release-key.keystore my_application.apk alias_name`

Note: If you are using IDE like Eclipse, the IDE will sign the APK file, and this step can be skipped.

Step 6:
Launch the emulator with the HTTP proxy option configured to route traffic to the proxy tool. Install the application and start it. The tester can then view HTTPS traffic via the proxy tool and can intercept it as well.

Command:
```
emulator -partition-size 128 -avd <AVD name> -http-proxy http://<proxy address>:<port>
```

javax.net.ssl.HttpsURLConnection

Step 1:
Create the classes listed below:

- httpsurlbypass.java
- bypass.java

Refer to Section 1 of the appendix for the code.

Step 2:
Add these classes to the existing source code in package `com.android`.

Step 3:
- Add `httpsurlbypass.trustAllHosts();` before getting an instance of `HttpsURLConnection`.
- Add `bypass.httpsurlconnectionbypass(<place the variable name of instance of HttpsURLConnection>);` before calling `getInputStream()`

Example:
```
    httpsurlbypass.trustAllHosts();
    HttpsURLConnection https = (HttpsURLConnection) url.openConnection();
    https.setHostnameVerifier(httpsurlbypass.hostname());
    https.getInputStream();
```

Step 4:
Compile the project to generate the installable.

Step 5:
Digitally sign the application using the following commands:

```
keytool –genk ey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA
-keysize 2048 -validity 10000
jarsigner -verbose -keystore my-release-key.keystore my_application.apk alias_name
```

Note: If you are using IDE like Eclipse, the IDE will sign the APK file, and this step can be skipped.

Step 6:
Launch the emulator with the HTTP proxy option configured to route traffic to the proxy tool. Install the application and start it. The tester can view HTTPS traffic via the proxy tool and can intercept it as well.

Command:

```
emulator -partition-size 128 -avd <AVD name> -http-proxy http://<proxy address>:<port>
```

android.webkit.WebView

Step 1:
Create the class listed below:

- bypass1.java

Refer to Section 1 of the appendix for the code.

Step 2:
Add this class to the existing source code in package: `com.android`.

Step 3:
Invoke the line of code below before calling the `loadurl` function:

```
    bypass1. webviewbypass(<place the variable name of instance of WebView>);
```

Example:

```
    bypass1.webviewbypass(mWebView); //mWebView is an instance of WebView class
    mWebView.loadUrl("https://mail.google.com");
```

Step 4:
Compile the project to generate the installable.

Step 5:
Digitally sign the application using the following commands:

- `keytool –genk ey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000`
- `jarsigner -verbose -keystore my-release-key.keystore my_application.apk alias_name`

Note: If you are using IDE like Eclipse, the IDE will sign the APK file, and this step can be skipped.

Step 6:
Launch the emulator with the HTTP proxy option configured to route traffic to the proxy tool. Install the application and start it. The tester can view HTTPS traffic via the proxy tool and can intercept it as well.

Command:

```
emulator -partition-size 128 -avd <AVD name> -http-proxy http://<proxy address>:<port>
```

Defeating SSL Certificate Validation Without the Application Source Code

Now that we know how to defeat SSL certificate validation if the source code is available, let's discuss the scenario where source code is not available and only the compiled installable is provided. In this case, we always have the option to decompile the code. When we decompile an Android installable file, we get Dalvik code, which is human readable and editable. Having knowledge of Dalvik code helps the penetration tester edit an Android installable and perform reverse engineering. Refer to http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html for information about Dalvik Opcode.

The code to be inserted into the installable can be developed by comparing the decompiled code of the installables of two sample applications—one with SSL certificate bypass code and the other without it.

org.apache.http.impl.client.DefaultHttpClient

To insert code into the installable, we need a sample project created. So, create a sample application HelloWebView (http://developer.android.com/resources/tutorials/views/hello-webview.html), and replace the contents of HelloWebView. Java class with the contents below.

```
package com.android;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;


public class HelloWebView extends Activity {
    /** Called when the activity is first created. */
 WebView mWebView;


    @Override
 public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);


     try {
   web();
  } catch (IOException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
   mWebView = (WebView) findViewById(R.id.webview);
   mWebView.loadData("Fail Traffic was not routed via proxy tool", "text/html", "utf-
8");
  } catch (Throwable e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
   mWebView = (WebView) findViewById(R.id.webview);
   mWebView.loadData("Fail Traffic was not routed via proxy tool", "text/html", "utf-
8");
```

```
   }
  }

    public void web() throws ClientProtocolException, IOException
    {
    mWebView = (WebView) findViewById(R.id.webview);
    mWebView.getSettings().setJavaScriptEnabled(true);
    DefaultHttpClient client = new DefaultHttpClient();
    HttpGet get = new HttpGet("https://mail.google.com");
    HttpResponse getResponse = client.execute(get);
    if(getResponse.getStatusLine().toString().equals("HTTP/1.1 200 OK"))
    {
      mWebView.loadData("Success Traffic was routed via proxy tool", "text/html", "utf-
8");
    }
  }
}
```

Step 1:
Compile the application.

Step 2:
Decompile the installable available at `<root path of project>/bin` using the command:

```
apktool.bat d -d <path of installable followed by installable file name> <output folder>
```

Step 3:
Create a folder `<root path where decompiled code is available>/com/Android` if it does not exist.

Step 4:
Create the files listed below within the `<root path where decompiled code is available>/com/Android` folder.

- EasySSLSocketFactory.java
- EasyX509TrustManager.java
- MyHttpClient.java

Refer to Section 2 of the appendix for the code.

Step 5:
Replace following lines of code in HelloWebView.Java.

```
new-instance v0, Lorg/apache/http/impl/client/DefaultHttpClient;

#v0=(UninitRef);
invoke-direct {v0}, Lorg/apache/http/impl/client/DefaultHttpClient;-><init>()V
```

with

```
new-instance v0, Lcom/android/MyHttpClient;

#v0=(UninitRef);
invoke-direct {v0}, Lcom/android/MyHttpClient;-><init>()V
```

Note: Replace the v0 register with the correct register for the installable that is decompiled and being edited during real time. It can be identified by reading the decompiled code.

Step 6:

Compile the code using the following command:

```
apktool b –d <folder containing decompiled code>
```

Step 7:

Digitally sign the application using the following commands:

- `keytool –genk ey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000`
- `jarsigner -verbose -keystore my-release-key.keystore my_application.apk alias_name`

Step 8:

Launch the emulator with the HTTP proxy setting configured to route traffic to the proxy tool. Install the application and launch it. HTTPS traffic will be routed via the proxy tool.

Command:

```
emulator -partition-size 128 -avd <AVD name> -http-proxy http://<proxy address>:<port>
```

javax.net.ssl.HttpsURLConnection

In this case, we replace the contents of HelloWebView.Java class with the contents below:

```
package com.android;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import javax.net.ssl.HttpsURLConnection;
import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;
import java.net.MalformedURLException;
import java.net.URL;

public class test1234 extends Activity {
    /** Called when the activity is first created. */
 WebView mWebView;
 String response;
 @Override
 public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
     mWebView = (WebView) findViewById(R.id.webview);
        mWebView.getSettings().setJavaScriptEnabled(true);

     URL url;
  try {
  //httpsurlbypass.trustAllHosts();
  url = new URL("https://mail.google.com");
     HttpsURLConnection https = (HttpsURLConnection) url.openConnection();
  //bypass.httpsurlconnectionbypass(https);
  https.getInputStream();
  mWebView.loadData("Success Traffic was  routed via proxy tool", "text/html", "utf-
8");
  }
  catch (MalformedURLException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
   mWebView.loadData("Fail Traffic was not routed via proxy tool", "text/html", "utf-
8");
  } catch (IOException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
   mWebView.loadData("Fail Traffic was not routed via proxy tool", "text/html", "utf-
8");
  }
 }
}
```

Step 1:
Compile the application.

Step 2:
Decompile the installable available at `<root path of project>/bin` using the command:

```
apktool.bat d –d < path of installable followed by installable file name> <output folder>
```

Step 3:
Create a folder `<root path where decompiled code is available>/com/Android` if it does not exist.

Step 4:
Add the classes listed below to the `<root path where decompiled code is available>/com/Android folder.`

- bypass.java
- httpsurlbypass.java
- httpsurlbypass$1.java
- httpsurlbypass$2.java

Refer to Section 2 of the appendix for the code.

Step 5:
Modify HelloWebView.Java as described below.

Add `invoke-static {}, Lcom/android/httpsurlbypass;->trustAllHosts()V` before getting an instance of `HttpsURLConnection` that is before the following:

```
#v3=(Reference);
invoke-direct {v2, v3}, Ljava/net/URL;-><init>(Ljava/lang/String;)V
```

Add `invoke-static {v1}, Lcom/android/bypass;->httpsurlconnectionbypass(Ljavax/net/ssl/HttpsURLConnection;)V` after the instance of `HttpsURLConnection` is created that is after `.local v1, https:Ljavax/net/ssl/HttpsURLConnection;`

Note: Replace the v2 and v3 register with the correct register for the installable that is decompiled and being edited during real time. It can be identified by reading the decompiled code.

Step 6:
Compile the code using the following command:

```
apktool b –d <folder containing decompiled code>
```

Step 7:

Digitally sign the application using the following commands:

- `keytool –genk ey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000`
- `jarsigner -verbose -keystore my-release-key.keystore my_application.apk alias_name`

Step 8:

Launch the emulator with the HTTP proxy setting configured to route traffic to the proxy tool. Install the application and launch it. HTTPS traffic will be routed via the proxy tool.

Command:

`emulator -partition-size 128 -avd <AVD name> -http-proxy` http://<proxy address>:<port>

android.webkit.WebView

In this case, we replace `mWebView.loadUrl("http://www.google.com");` in HelloWebView.Java class with `mWebView.loadUrl("https://mail.google.com");`

Step 1:

Compile the application.

Step 2:

Decompile the installable available at `<root path of project>/bin` using the command:

```
apktool.bat d –d < path of installable followed by installable file name> <output
folder>
```

Step 3:

Create a folder `<root path where decompiled code is available>/com/Android` if it does not exist.

Step 4:

Add the following classes to the `<root path where decompiled code is available>/com/Android` folder.

- bypass1.java
- bypass1$1.java

Refer to Section 2 of the appendix for the code.

Step 5:

Add the following lines of code to HelloWebView.Java

```
    iget-object v0, p0, Lcom/android/HelloWebView;->mWebView:Landroid/webkit/WebView;
    invoke-static {v0}, Lcom/android/bypass1;->webviewbypass(Landroid/webkit/WebView;)V
```

before invoking loadurl function that is before:

```
    iget-object v0, p0, Lcom/android/HelloWebView;->mWebView:Landroid/webkit/WebView;
    const-string v1, "https://mail.google.com"
    #v1=(Reference);
    invoke-virtual {v0, v1}, Landroid/webkit/WebView;->loadUrl(Ljava/lang/String;)V
```

Note: Replace the `v0` and `v1` register with the correct register for the installable that is decompiled and being edited during real time. It can be identified by reading the decompiled code.

Step 6:

Compile the code using the following command:

```
apktool b –d <folder containing decompiled code>
```

Step 7:

Digitally sign the application using the following commands:

- `keytool –genk ey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000`
- `jarsigner -verbose -keystore my-release-key.keystore my_application.apk alias_name`

Step 8:

Launch the emulator with the HTTP proxy settings configured to route traffic to the proxy tool. Install the application and launch it. HTTPS traffic will be routed via the proxy tool.

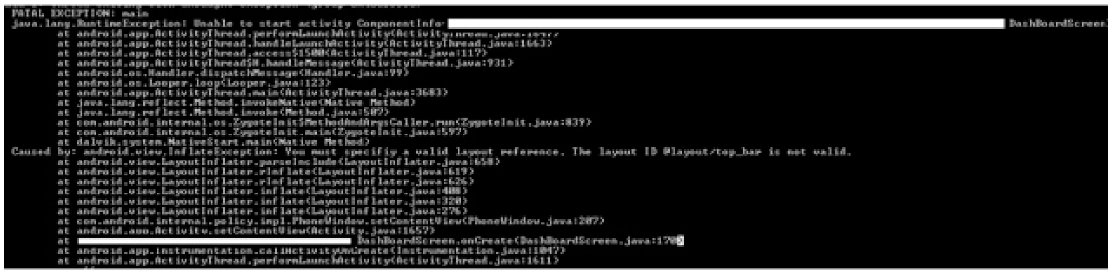
Command:

```
emulator -partition-size 128 -avd <AVD name> -http-proxy http://<proxy address>:<port>
```

Pro Tips

Here are a few tips that would be useful when testing Android applications and going through the steps listed in this paper:

1. An APK file is an archive file, and it can be opened using archiving tools such as WinRAR.
2. Sometimes the application does not work when we recompile the decompiled code. The error thrown is similar to the one shown in the image below.



```
FATAL EXCEPTION: main
java.lang.RuntimeException: Unable to start activity ComponentInfo{[redacted]DashBoardScreen
        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:1577)
        at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:1663)
        at android.app.ActivityThread.access$1500(ActivityThread.java:117)
        at android.app.ActivityThread$H.handleMessage(ActivityThread.java:931)
        at android.os.Handler.dispatchMessage(Handler.java:99)
        at android.os.Looper.loop(Looper.java:123)
        at android.app.ActivityThread.main(ActivityThread.java:3683)
        at java.lang.reflect.Method.invokeNative(Native Method)
        at java.lang.reflect.Method.invoke(Method.java:507)
        at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:839)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:597)
        at dalvik.system.NativeStart.main(Native Method)
Caused by: android.view.InflateException: You must specifiy a valid layout reference. The layout ID @layout/top_bar is not valid.
        at android.view.LayoutInflater.parseInclude(LayoutInflater.java:658)
        at android.view.LayoutInflater.rInflate(LayoutInflater.java:619)
        at android.view.LayoutInflater.rInflate(LayoutInflater.java:626)
        at android.view.LayoutInflater.inflate(LayoutInflater.java:408)
        at android.view.LayoutInflater.inflate(LayoutInflater.java:320)
        at android.view.LayoutInflater.inflate(LayoutInflater.java:276)
        at com.android.internal.policy.impl.PhoneWindow.setContentView(PhoneWindow.java:207)
        at android.app.Activity.setContentView(Activity.java:1657)
        at [redacted]DashBoardScreen.onCreate(DashBoardScreen.java:170)
        at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1047)
        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:1611)
```

Figure 2. Error after recompiling decompiled code.

There is a workaround for this. Unzip all the contents of the APK file obtained after compiling the recompiled code. Replace the "res" folder and its contents with that of the original APK file. Then recreate the archive file with the extension of APK. Install this new APK file. The application should then run without this error.

3. While inserting extra Dalvik code into decompiled code, look to see if the developer has created wrapper classes that in turn use the classes described above to access the web. If that is the case, you would only need to modify that wrapper class.

Summary

This paper provides a method by which penetration testers can intercept HTTPS traffic while performing security testing of Android applications. Once HTTPS traffic is intercepted, security testing of Android applications is similar to web application penetration testing. This can help in addressing application security issues before the application is released to the public.

Appendix

Section 1

EasyX509TrustManager.java (http://stackoverflow.com/questions/16414396/is-this-https-connection-secure)

```
package com.android;

import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;

public class EasyX509TrustManager implements X509TrustManager {

    private X509TrustManager standardTrustManager = null;

    /**
     * Constructor for EasyX509TrustManager.
     */
    public EasyX509TrustManager(KeyStore keystore) throws NoSuchAlgorithmException,
KeyStoreException {
        super();
        TrustManagerFactory factory = TrustManagerFactory.
getInstance(TrustManagerFactory.getDefaultAlgorithm());
        factory.init(keystore);
        TrustManager[] trustmanagers = factory.getTrustManagers();
        if (trustmanagers.length == 0) {
            throw new NoSuchAlgorithmException("no trust manager found");
        }
        this.standardTrustManager = (X509TrustManager) trustmanagers[0];
    }

    /**
     * @see javax.net.ssl.X509TrustManager#checkClientTrusted(X509Certificate[],String
authType)
     */
    public void checkClientTrusted(X509Certificate[] certificates, String authType)
throws CertificateException {
        standardTrustManager.checkClientTrusted(certificates, authType);
    }

    /**
     * @see javax.net.ssl.X509TrustManager#checkServerTrusted(X509Certificate[],String
authType)
     */
    public void checkServerTrusted(X509Certificate[] certificates, String authType)
throws CertificateException {
        if ((certificates != null) && (certificates.length == 1)) {
            certificates[0].checkValidity();
        } else {
            standardTrustManager.checkServerTrusted(certificates, authType);
        }
    }

    /**
     * @see javax.net.ssl.X509TrustManager#getAcceptedIssuers()
     */
    public X509Certificate[] getAcceptedIssuers() {
        return this.standardTrustManager.getAcceptedIssuers();
    }

}
```

EasySSLSocketFactory.java (http://stackoverflow.com/questions/16414396/is-this-https-connection-secure**)**

```
package com.android;

import java.io.IOException;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.UnknownHostException;

import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.TrustManager;

import org.apache.http.conn.ConnectTimeoutException;
import org.apache.http.conn.scheme.LayeredSocketFactory;
import org.apache.http.conn.scheme.SocketFactory;
import org.apache.http.params.HttpConnectionParams;
import org.apache.http.params.HttpParams;

/**
 * This socket factory will create ssl socket that accepts self-signed certificate.
 *
 * @author olamy
 * @version $Id: EasySSLSocketFactory.java 765355 2009-04-15 20:59:07Z evenisse $
 * @since 1.2.3
 */
public class EasySSLSocketFactory implements SocketFactory, LayeredSocketFactory {

 private SSLContext sslcontext = null;

 private static SSLContext createEasySSLContext() throws IOException {
  try {
   SSLContext context = SSLContext.getInstance("TLS");
   context.init(null, new TrustManager[] {  new EasyX509TrustManager(null) }, null);
   return context;
  } catch (Exception e) {
   throw new IOException(e.getMessage());
  }
 }

 private SSLContext getSSLContext() throws IOException {
  if (this.sslcontext == null) {
   this.sslcontext = createEasySSLContext();
```

```
    }
    return this.sslcontext;
  }

  /**
   * @see org.apache.http.conn.scheme.SocketFactory#connectSocket(java.net.Socket, java.
lang.String, int,
   *      java.net.InetAddress, int, org.apache.http.params.HttpParams)
   */
  public Socket connectSocket(Socket sock, String host, int port, InetAddress
localAddress, int localPort,
     HttpParams params) throws IOException, UnknownHostException, ConnectTimeoutException
{
   int connTimeout = HttpConnectionParams.getConnectionTimeout(params);
   int soTimeout = HttpConnectionParams.getSoTimeout(params);
   InetSocketAddress remoteAddress = new InetSocketAddress(host, port);
   SSLSocket sslsock = (SSLSocket) ((sock != null) ? sock : createSocket());

   if ((localAddress != null) || (localPort > 0)) {
    // we need to bind explicitly
    if (localPort < 0) {
     localPort = 0; // indicates "any"
    }
    InetSocketAddress isa = new InetSocketAddress(localAddress, localPort);
    sslsock.bind(isa);
   }

   sslsock.connect(remoteAddress, connTimeout);
   sslsock.setSoTimeout(soTimeout);
   return sslsock;

  }

  /**
   * @see org.apache.http.conn.scheme.SocketFactory#createSocket()
   */
  public Socket createSocket() throws IOException {
   return getSSLContext().getSocketFactory().createSocket();
  }

  /**
   * @see org.apache.http.conn.scheme.SocketFactory#isSecure(java.net.Socket)
   */
  public boolean isSecure(Socket socket) throws IllegalArgumentException {
   return true;
  }

  /**
   * @see org.apache.http.conn.scheme.LayeredSocketFactory#createSocket(java.net.Socket,
java.lang.String, int,
   *      boolean)
   */
  public Socket createSocket(Socket socket, String host, int port, boolean autoClose)
throws IOException,
    UnknownHostException {
   return getSSLContext().getSocketFactory().createSocket(socket, host, port,
autoClose);
  }

  // -------------------------------------------------------------------
  // javadoc in org.apache.http.conn.scheme.SocketFactory says :
  // Both Object.equals() and Object.hashCode() must be overridden
  // for the correct operation of some connection managers
  // -------------------------------------------------------------------

  public boolean equals(Object obj) {
   return ((obj != null) && obj.getClass().equals(EasySSLSocketFactory.class));
  }

  public int hashCode() {
   return EasySSLSocketFactory.class.hashCode();
  }

}
```

MyHttpClient.java

```
package com.android;

import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.PlainSocketFactory;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.conn.SingleClientConnManager;
import org.apache.http.params.HttpParams;


public class MyHttpClient extends DefaultHttpClient {

 //This is the constructor
    public MyHttpClient()
    {

    }
//use this constructor if parameters needs to be sent
    public MyHttpClient(ClientConnectionManager con,HttpParams param)
    {
     setParams(param);

    }

//overriding the function to skip ssl certificate validation
    @Override
    protected ClientConnectionManager createClientConnectionManager() {
     SchemeRegistry registry = new SchemeRegistry();
        registry.register(
            new Scheme("http", PlainSocketFactory.getSocketFactory(), 80));
        registry.register(new Scheme("https", new EasySSLSocketFactory(), 443));
        return new SingleClientConnManager(getParams(), registry);

    }
 }
```

httpsurlbypass.java

```
package com.android;

import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

public class httpsurlbypass
{
 public static void trustAllHosts() {
     // Create a trust manager that does not validate certificate chains
  TrustManager[] trustAllCerts = new TrustManager[] { new X509TrustManager()
  {public java.security.cert.X509Certificate[] getAcceptedIssuers() {return new java.
security.cert.X509Certificate[] {};}
  public void checkClientTrusted(X509Certificate[] chain,String authType) throws
CertificateException {                 }
  public void checkServerTrusted(X509Certificate[] chain,String authType) throws
CertificateException {                 } } };
  // Install the all-trusting trust manager
  try {
  SSLContext sc = SSLContext.getInstance("TLS");
  sc.init(null, trustAllCerts, new java.security.SecureRandom());
  HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());         }
  catch (Exception e)
  {
  e.printStackTrace();
  }
  }

 public static HostnameVerifier hostname()
 {
  return new HostnameVerifier() {
   public boolean verify(String hostname, SSLSession session) {
    return true;
    } };
 }
}
```

bypass.java
```
package com.android;

import javax.net.ssl.HttpsURLConnection;

import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.net.http.*;
import android.webkit.SslErrorHandler;

public class bypass
{

 public static void httpsurlconnectionbypass(HttpsURLConnection https)
 {
  https.setHostnameVerifier(httpsurlbypass.hostname());
 }

}
```

bypass1.java
```
package com.android;

import javax.net.ssl.HttpsURLConnection;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.net.http.*;
import android.webkit.SslErrorHandler;

public class bypass1
{

 public static WebViewClient byp()
 {
  return new WebViewClient(){
      public void onReceivedSslError(WebView view, SslErrorHandler handler, SslError
error){
      handler.proceed();
      }
      };
 }

 public static void webviewbypass(WebView view)
 {
  view.setWebViewClient(byp());
 }

}
```

Section 2

EasySSLSocketFactory.java

```
package com.android; class EasySSLSocketFactory {/*

.class public Lcom/android/EasySSLSocketFactory;
.super Ljava/lang/Object;
.source “EasySSLSocketFactory.java”

# interfaces
.implements Lorg/apache/http/conn/scheme/SocketFactory;
.implements Lorg/apache/http/conn/scheme/LayeredSocketFactory;


# instance fields
.field private sslcontext:Ljavax/net/ssl/SSLContext;


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

    .prologue
    .line 26
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    .line 28
    #p0=(Reference);
    const/4 v0, 0x0

    #v0=(Null);
    iput-object v0, p0, Lcom/android/EasySSLSocketFactory;->sslcontext:Ljavax/net/ssl/
SSLContext;

    .line 26
    return-void
.end method

.method private static createEasySSLContext()Ljavax/net/ssl/SSLContext;
    .locals 7
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/io/IOException;
        }
    .end annotation

    .prologue
    .line 32
    :try_start_0
    const-string v2, “TLS”

    #v2=(Reference);
    invoke-static {v2}, Ljavax/net/ssl/SSLContext;->getInstance(Ljava/lang/String;)
Ljavax/net/ssl/SSLContext;
```

```
    move-result-object v0

    .line 33
    .local v0, context:Ljavax/net/ssl/SSLContext;
    #v0=(Reference);
    const/4 v2, 0x0

    #v2=(Null);
    const/4 v3, 0x1

    #v3=(One);
    new-array v3, v3, [Ljavax/net/ssl/TrustManager;

    #v3=(Reference);
    const/4 v4, 0x0

    #v4=(Null);
    new-instance v5, Lcom/android/EasyX509TrustManager;

    #v5=(UninitRef);
    const/4 v6, 0x0

    #v6=(Null);
    invoke-direct {v5, v6}, Lcom/android/EasyX509TrustManager;-><init>(Ljava/security/
KeyStore;)V

    #v5=(Reference);
    aput-object v5, v3, v4

    const/4 v4, 0x0

    invoke-virtual {v0, v2, v3, v4}, Ljavax/net/ssl/SSLContext;->init([Ljavax/net/ssl/
KeyManager;[Ljavax/net/ssl/TrustManager;Ljava/security/SecureRandom;)V
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    .line 34
    return-object v0

    .line 35
    .end local v0           #context:Ljavax/net/ssl/SSLContext;
    :catch_0
    move-exception v2

    #v2=(Reference);
    move-object v1, v2

    .line 36
    .local v1, e:Ljava/lang/Exception;
    #v1=(Reference);
    new-instance v2, Ljava/io/IOException;

    #v2=(UninitRef);
    invoke-virtual {v1}, Ljava/lang/Exception;->getMessage()Ljava/lang/String;
    move-result-object v3

    #v3=(Reference);
    invoke-direct {v2, v3}, Ljava/io/IOException;-><init>(Ljava/lang/String;)V

    #v2=(Reference);
    throw v2
.end method

.method private getSSLContext()Ljavax/net/ssl/SSLContext;
    .locals 1
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/io/IOException;
        }
    .end annotation

    .prologue
    .line 41
    iget-object v0, p0, Lcom/android/EasySSLSocketFactory;->sslcontext:Ljavax/net/ssl/
SSLContext;

    #v0=(Reference);
    if-nez v0, :cond_0

    .line 42
    invoke-static {}, Lcom/android/EasySSLSocketFactory;->createEasySSLContext()Ljavax/
net/ssl/SSLContext;

    move-result-object v0

    iput-object v0, p0, Lcom/android/EasySSLSocketFactory;->sslcontext:Ljavax/net/ssl/
SSLContext;

    .line 44
    :cond_0
    iget-object v0, p0, Lcom/android/EasySSLSocketFactory;->sslcontext:Ljavax/net/ssl/
SSLContext;

    return-object v0
.end method


# virtual methods
.method public connectSocket(Ljava/net/Socket;Ljava/lang/String;ILjava/net/
InetAddress;ILorg/apache/http/params/HttpParams;)Ljava/net/Socket;
    .locals 6
    .parameter “sock”
    .parameter “host”
    .parameter “port”
    .parameter “localAddress”
    .parameter “localPort”
    .parameter “params”
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/io/IOException;,
            Ljava/net/UnknownHostException;,
            Lorg/apache/http/conn/ConnectTimeoutException;
        }
    .end annotation

    .prologue
    .line 53
    invoke-static {p6}, Lorg/apache/http/params/HttpConnectionParams;-
>getConnectionTimeout(Lorg/apache/http/params/HttpParams;)I

    move-result v0

    .line 54
    .local v0, connTimeout:I
    #v0=(Integer);
    invoke-static {p6}, Lorg/apache/http/params/HttpConnectionParams;-
>getSoTimeout(Lorg/apache/http/params/HttpParams;)I

    move-result v3

    .line 55
    .local v3, soTimeout:I
    #v3=(Integer);
    new-instance v2, Ljava/net/InetSocketAddress;

    #v2=(UninitRef);
    invoke-direct {v2, p2, p3}, Ljava/net/InetSocketAddress;-><init>(Ljava/lang/
String;I)V

    .line 56
    .local v2, remoteAddress:Ljava/net/InetSocketAddress;
    #v2=(Reference);
    if-eqz p1, :cond_3

    move-object v4, p1

    :goto_0
    #v4=(Reference);v5=(Conflicted);
    check-cast v4, Ljavax/net/ssl/SSLSocket;

    .line 58
    .local v4, sslsock:Ljavax/net/ssl/SSLSocket;
    if-nez p4, :cond_0

    if-lez p5, :cond_2

    .line 60
    :cond_0
    if-gez p5, :cond_1

    .line 61
    const/4 p5, 0x0

    .line 63
    :cond_1
    new-instance v1, Ljava/net/InetSocketAddress;
    #v1=(UninitRef);
    invoke-direct {v1, p4, p5}, Ljava/net/InetSocketAddress;-><init>(Ljava/net/
InetAddress;I)V

    .line 64
    .local v1, isa:Ljava/net/InetSocketAddress;
    #v1=(Reference);
    invoke-virtual {v4, v1}, Ljavax/net/ssl/SSLSocket;->bind(Ljava/net/SocketAddress;)V

    .line 67
    .end local v1           #isa:Ljava/net/InetSocketAddress;
    :cond_2
    #v1=(Conflicted);
    invoke-virtual {v4, v2, v0}, Ljavax/net/ssl/SSLSocket;->connect(Ljava/net/
SocketAddress;I)V

    .line 68
    invoke-virtual {v4, v3}, Ljavax/net/ssl/SSLSocket;->setSoTimeout(I)V

    .line 69
    return-object v4

    .line 56
    .end local v4           #sslsock:Ljavax/net/ssl/SSLSocket;
    :cond_3
    #v1=(Uninit);v4=(Uninit);v5=(Uninit);
    invoke-virtual {p0}, Lcom/android/EasySSLSocketFactory;->createSocket()Ljava/net/
Socket;

    move-result-object v5

    #v5=(Reference);
    move-object v4, v5

    #v4=(Reference);
    goto :goto_0
.end method

.method public createSocket()Ljava/net/Socket;
    .locals 1
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/io/IOException;
        }
    .end annotation

    .prologue
    .line 77
    invoke-direct {p0}, Lcom/android/EasySSLSocketFactory;->getSSLContext()Ljavax/net/
ssl/SSLContext;

    move-result-object v0
    #v0=(Reference);
    invoke-virtual {v0}, Ljavax/net/ssl/SSLContext;->getSocketFactory()Ljavax/net/ssl/
SSLSocketFactory;

    move-result-object v0

    invoke-virtual {v0}, Ljavax/net/ssl/SSLSocketFactory;->createSocket()Ljava/net/
Socket;

    move-result-object v0

    return-object v0
.end method

.method public createSocket(Ljava/net/Socket;Ljava/lang/String;IZ)Ljava/net/Socket;
    .locals 1
    .parameter “socket”
    .parameter “host”
    .parameter “port”
    .parameter “autoClose”
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/io/IOException;,
            Ljava/net/UnknownHostException;
        }
    .end annotation

    .prologue
    .line 93
    invoke-direct {p0}, Lcom/android/EasySSLSocketFactory;->getSSLContext()Ljavax/net/
ssl/SSLContext;

    move-result-object v0

    #v0=(Reference);
    invoke-virtual {v0}, Ljavax/net/ssl/SSLContext;->getSocketFactory()Ljavax/net/ssl/
SSLSocketFactory;

    move-result-object v0

    invoke-virtual {v0, p1, p2, p3, p4}, Ljavax/net/ssl/SSLSocketFactory;-
>createSocket(Ljava/net/Socket;Ljava/lang/String;IZ)Ljava/net/Socket;

    move-result-object v0

    return-object v0
.end method

.method public equals(Ljava/lang/Object;)Z
    .locals 2
    .parameter “obj”

    .prologue
    .line 103
```

```
        if-eqz p1, :cond_0

        invoke-virtual {p1}, Ljava/lang/Object;->getClass()Ljava/lang/Class;

        move-result-object v0

        #v0=(Reference);
        const-class v1, Lcom/android/EasySSLSocketFactory;

        #v1=(Reference);
        invoke-virtual {v0, v1}, Ljava/lang/Object;->equals(Ljava/lang/Object;)Z

        move-result v0

        #v0=(Boolean);
        if-eqz v0, :cond_0

        const/4 v0, 0x1

        :goto_0
        #v1=(Conflicted);
        return v0

        :cond_0
        #v0=(Conflicted);
        const/4 v0, 0x0

        #v0=(Null);
        goto :goto_0
.end method

.method public hashCode()I
        .locals 1

        .prologue
        .line 107
        const-class v0, Lcom/android/EasySSLSocketFactory;

        #v0=(Reference);
        invoke-virtual {v0}, Ljava/lang/Object;->hashCode()I

        move-result v0

        #v0=(Integer);
        return v0
.end method

.method public isSecure(Ljava/net/Socket;)Z
        .locals 1
        .parameter "socket"
        .annotation system Ldalvik/annotation/Throws;
            value = {
                Ljava/lang/IllegalArgumentException;
            }
```

```
    .end annotation

    .prologue
    .line 84
    const/4 v0, 0x1

    #v0=(One);
    return v0
.end method

*/}
```

EasyX509TrustManager.java

```
package com.android; class EasyX509TrustManager {/*

.class public Lcom/android/EasyX509TrustManager;
.super Ljava/lang/Object;
.source "EasyX509TrustManager.java"

# interfaces
.implements Ljavax/net/ssl/X509TrustManager;


# instance fields
.field private standardTrustManager:Ljavax/net/ssl/X509TrustManager;


# direct methods
.method public constructor <init>(Ljava/security/KeyStore;)V
    .locals 4
    .parameter "keystore"
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/security/NoSuchAlgorithmException;,
            Ljava/security/KeyStoreException;
        }
    .end annotation

    .prologue
    .line 19
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    .line 13
    #p0=(Reference);
    const/4 v2, 0x0

    #v2=(Null);
    iput-object v2, p0, Lcom/android/EasyX509TrustManager;-
>standardTrustManager:Ljavax/net/ssl/X509TrustManager;

    .line 20
    invoke-static {}, Ljavax/net/ssl/TrustManagerFactory;->getDefaultAlgorithm()Ljava/
lang/String;
```

```
    move-result-object v2

    #v2=(Reference);
    invoke-static {v2}, Ljavax/net/ssl/TrustManagerFactory;->getInstance(Ljava/lang/
String;)Ljavax/net/ssl/TrustManagerFactory;

    move-result-object v0

    .line 21
    .local v0, factory:Ljavax/net/ssl/TrustManagerFactory;
    #v0=(Reference);
    invoke-virtual {v0, p1}, Ljavax/net/ssl/TrustManagerFactory;->init(Ljava/security/
KeyStore;)V

    .line 22
    invoke-virtual {v0}, Ljavax/net/ssl/TrustManagerFactory;->getTrustManagers()
[Ljavax/net/ssl/TrustManager;

    move-result-object v1

    .line 23
    .local v1, trustmanagers:[Ljavax/net/ssl/TrustManager;
    #v1=(Reference);
    array-length v2, v1

    #v2=(Integer);
    if-nez v2, :cond_0

    .line 24
    new-instance v2, Ljava/security/NoSuchAlgorithmException;

    #v2=(UninitRef);
    const-string v3, “no trust manager found”

    #v3=(Reference);
    invoke-direct {v2, v3}, Ljava/security/NoSuchAlgorithmException;-><init>(Ljava/
lang/String;)V

    #v2=(Reference);
    throw v2

    .line 26
    :cond_0
    #v2=(Integer);v3=(Uninit);
    const/4 v2, 0x0

    #v2=(Null);
    aget-object v2, v1, v2

    check-cast v2, Ljavax/net/ssl/X509TrustManager;

    #v2=(Reference);
    iput-object v2, p0, Lcom/android/EasyX509TrustManager;-
>standardTrustManager:Ljavax/net/ssl/X509TrustManager;
```

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


# virtual methods
.method public checkClientTrusted([Ljava/security/cert/X509Certificate;Ljava/lang/
String;)V
    .locals 1
    .parameter "certificates"
    .parameter "authType"
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/security/cert/CertificateException;
        }
    .end annotation

    .prologue
    .line 33
    iget-object v0, p0, Lcom/android/EasyX509TrustManager;-
>standardTrustManager:Ljavax/net/ssl/X509TrustManager;

    #v0=(Reference);
    invoke-interface {v0, p1, p2}, Ljavax/net/ssl/X509TrustManager;-
>checkClientTrusted([Ljava/security/cert/X509Certificate;Ljava/lang/String;)V

    .line 34
    return-void
.end method

.method public checkServerTrusted([Ljava/security/cert/X509Certificate;Ljava/lang/
String;)V
    .locals 2
    .parameter "certificates"
    .parameter "authType"
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/security/cert/CertificateException;
        }
    .end annotation

    .prologue
    .line 40
    if-eqz p1, :cond_0

    array-length v0, p1

    #v0=(Integer);
    const/4 v1, 0x1

    #v1=(One);
    if-ne v0, v1, :cond_0
    .line 41
    const/4 v0, 0x0

    #v0=(Null);
    aget-object v0, p1, v0

    invoke-virtual {v0}, Ljava/security/cert/X509Certificate;->checkValidity()V

    .line 45
    :goto_0
    #v0=(Reference);v1=(Conflicted);
    return-void

    .line 43
    :cond_0
    #v0=(Conflicted);
    iget-object v0, p0, Lcom/android/EasyX509TrustManager;-
>standardTrustManager:Ljavax/net/ssl/X509TrustManager;

    #v0=(Reference);
    invoke-interface {v0, p1, p2}, Ljavax/net/ssl/X509TrustManager;-
>checkServerTrusted([Ljava/security/cert/X509Certificate;Ljava/lang/String;)V

    goto :goto_0
.end method

.method public getAcceptedIssuers()[Ljava/security/cert/X509Certificate;
    .locals 1

    .prologue
    .line 51
    iget-object v0, p0, Lcom/android/EasyX509TrustManager;-
>standardTrustManager:Ljavax/net/ssl/X509TrustManager;

    #v0=(Reference);
    invoke-interface {v0}, Ljavax/net/ssl/X509TrustManager;->getAcceptedIssuers()
[Ljava/security/cert/X509Certificate;

    move-result-object v0

    return-object v0
.end method

*/}
```

MyHttpClient.java

```
package com.android; class MyHttpClient {/*

.class public Lcom/android/MyHttpClient;
.super Lorg/apache/http/impl/client/DefaultHttpClient;
.source "MyHttpClient.java"


# direct methods
```

```
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 15
    invoke-direct {p0}, Lorg/apache/http/impl/client/DefaultHttpClient;-><init>()V

    .line 19
    #p0=(Reference);
    return-void
.end method

.method public constructor <init>(Lorg/apache/http/conn/ClientConnectionManager;Lorg/
apache/http/params/HttpParams;)V
    .locals 0
    .parameter "con"
    .parameter "param"

    .prologue
    .line 20
    invoke-direct {p0}, Lorg/apache/http/impl/client/DefaultHttpClient;-><init>()V

    .line 22
    #p0=(Reference);
    invoke-virtual {p0, p2}, Lcom/android/MyHttpClient;->setParams(Lorg/apache/http/
params/HttpParams;)V

    .line 24
    return-void
.end method


# virtual methods
.method protected createClientConnectionManager()Lorg/apache/http/conn/
ClientConnectionManager;
    .locals 5

    .prologue
    .line 28
    new-instance v0, Lorg/apache/http/conn/scheme/SchemeRegistry;

    #v0=(UninitRef);
    invoke-direct {v0}, Lorg/apache/http/conn/scheme/SchemeRegistry;-><init>()V

    .line 30
    .local v0, registry:Lorg/apache/http/conn/scheme/SchemeRegistry;
    #v0=(Reference);
    new-instance v1, Lorg/apache/http/conn/scheme/Scheme;

    #v1=(UninitRef);
    const-string v2, "http"

    #v2=(Reference);
    invoke-static {}, Lorg/apache/http/conn/scheme/PlainSocketFactory;-
```

```
>getSocketFactory()Lorg/apache/http/conn/scheme/PlainSocketFactory;

    move-result-object v3

    #v3=(Reference);
    const/16 v4, 0x50

    #v4=(PosByte);
    invoke-direct {v1, v2, v3, v4}, Lorg/apache/http/conn/scheme/Scheme;-><init>(Ljava/
lang/String;Lorg/apache/http/conn/scheme/SocketFactory;I)V

    .line 29
    #v1=(Reference);
    invoke-virtual {v0, v1}, Lorg/apache/http/conn/scheme/SchemeRegistry;-
>register(Lorg/apache/http/conn/scheme/Scheme;)Lorg/apache/http/conn/scheme/Scheme;

    .line 31
    new-instance v1, Lorg/apache/http/conn/scheme/Scheme;

    #v1=(UninitRef);
    const-string v2, “https”

    new-instance v3, Lcom/android/EasySSLSocketFactory;

    #v3=(UninitRef);
    invoke-direct {v3}, Lcom/android/EasySSLSocketFactory;-><init>()V

    #v3=(Reference);
    const/16 v4, 0x1bb

    #v4=(PosShort);
    invoke-direct {v1, v2, v3, v4}, Lorg/apache/http/conn/scheme/Scheme;-><init>(Ljava/
lang/String;Lorg/apache/http/conn/scheme/SocketFactory;I)V

    #v1=(Reference);
    invoke-virtual {v0, v1}, Lorg/apache/http/conn/scheme/SchemeRegistry;-
>register(Lorg/apache/http/conn/scheme/Scheme;)Lorg/apache/http/conn/scheme/Scheme;

    .line 32
    new-instance v1, Lorg/apache/http/impl/conn/SingleClientConnManager;

    #v1=(UninitRef);
    invoke-virtual {p0}, Lcom/android/MyHttpClient;->getParams()Lorg/apache/http/
params/HttpParams;

    move-result-object v2

    invoke-direct {v1, v2, v0}, Lorg/apache/http/impl/conn/SingleClientConnManager;-
><init>(Lorg/apache/http/params/HttpParams;Lorg/apache/http/conn/scheme/
SchemeRegistry;)V

    #v1=(Reference);
    return-object v1
.end method
```

```
*/}
```

bypass.java

```
package com.android; class bypass {/*

.class public Lcom/android/bypass;
.super Ljava/lang/Object;
.source "bypass.java"


# direct methods
.method public constructor <init>()V
    .locals 0

     .prologue
    .line 6
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    #p0=(Reference);
    return-void
.end method

.method public static httpsurlconnectionbypass(Ljavax/net/ssl/HttpsURLConnection;)V
    .locals 1
    .parameter "http"

    .prologue
    .line 13
    invoke-static {}, Lcom/android/httpsurlbypass;->hostname()Ljavax/net/ssl/
HostnameVerifier;

    move-result-object v0

    #v0=(Reference);
    invoke-virtual {p0, v0}, Ljavax/net/ssl/HttpsURLConnection;-
>setHostnameVerifier(Ljavax/net/ssl/HostnameVerifier;)V

    .line 14
    return-void
.end method

*/}
```

httpsurlbypass.java

```
package com.android; class httpsurlbypass {/*

.class public Lcom/android/httpsurlbypass;
.super Ljava/lang/Object;
.source "httpsurlbypass.java"


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

```
    .locals 0

    .prologue
    .line 13
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    #p0=(Reference);
    return-void
.end method

.method public static hostname()Ljavax/net/ssl/HostnameVerifier;
    .locals 1

    .prologue
    .line 34
    new-instance v0, Lcom/android/httpsurlbypass$2;

    #v0=(UninitRef);
    invoke-direct {v0}, Lcom/android/httpsurlbypass$2;-><init>()V

    #v0=(Reference);
    return-object v0
.end method

.method public static trustAllHosts()V
    .locals 5

    .prologue
    .line 17
    const/4 v3, 0x1

    #v3=(One);
    new-array v2, v3, [Ljavax/net/ssl/TrustManager;

    #v2=(Reference);
    const/4 v3, 0x0

    #v3=(Null);
    new-instance v4, Lcom/android/httpsurlbypass$1;

    #v4=(UninitRef);
    invoke-direct {v4}, Lcom/android/httpsurlbypass$1;-><init>()V

    #v4=(Reference);
    aput-object v4, v2, v3

    .line 23
    .local v2, trustAllCerts:[Ljavax/net/ssl/TrustManager;
    :try_start_0
    const-string v3, "TLS"

    #v3=(Reference);
    invoke-static {v3}, Ljavax/net/ssl/SSLContext;->getInstance(Ljava/lang/String;)
Ljavax/net/ssl/SSLContext;
```

```
    move-result-object v1

    .line 24
    .local v1, sc:Ljavax/net/ssl/SSLContext;
    #v1=(Reference);
    const/4 v3, 0x0

    #v3=(Null);
    new-instance v4, Ljava/security/SecureRandom;

    #v4=(UninitRef);
    invoke-direct {v4}, Ljava/security/SecureRandom;-><init>()V

    #v4=(Reference);
    invoke-virtual {v1, v3, v2, v4}, Ljavax/net/ssl/SSLContext;->init([Ljavax/net/ssl/
KeyManager;[Ljavax/net/ssl/TrustManager;Ljava/security/SecureRandom;)V

    .line 25
    invoke-virtual {v1}, Ljavax/net/ssl/SSLContext;->getSocketFactory()Ljavax/net/ssl/
SSLSocketFactory;

    move-result-object v3

    #v3=(Reference);
    invoke-static {v3}, Ljavax/net/ssl/HttpsURLConnection;->setDefaultSSLSocketFactory(
Ljavax/net/ssl/SSLSocketFactory;)V
    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0

    .line 30
    .end local v1          #sc:Ljavax/net/ssl/SSLContext;
    :goto_0
    #v0=(Conflicted);v1=(Conflicted);v4=(Conflicted);
    return-void

    .line 26
    :catch_0
    #v0=(Uninit);
    move-exception v3

    move-object v0, v3

    .line 28
    .local v0, e:Ljava/lang/Exception;
    #v0=(Reference);
    invoke-virtual {v0}, Ljava/lang/Exception;->printStackTrace()V

    goto :goto_0
.end method

*/}
```

httpsurlbypass$1.java

```
package com.android; class httpsurlbypass$1 {/*

.class final Lcom/android/httpsurlbypass$1;
.super Ljava/lang/Object;
.source “httpsurlbypass.java”

# interfaces
.implements Ljavax/net/ssl/X509TrustManager;


# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/android/httpsurlbypass;->trustAllHosts()V
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = null
.end annotation


# direct methods
.method constructor <init>()V
    .locals 0

    .prologue
    .line 17
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    .line 1
    #p0=(Reference);
    return-void
.end method


# virtual methods
.method public checkClientTrusted([Ljava/security/cert/X509Certificate;Ljava/lang/
String;)V
    .locals 0
    .parameter “chain”
    .parameter “authType”
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/security/cert/CertificateException;
        }
    .end annotation

    .prologue
    .line 19
    return-void
.end method

.method public checkServerTrusted([Ljava/security/cert/X509Certificate;Ljava/lang/
String;)V
    .locals 0
    .parameter "chain"
    .parameter "authType"
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/security/cert/CertificateException;
        }
    .end annotation

    .prologue
    .line 20
    return-void
.end method

.method public getAcceptedIssuers()[Ljava/security/cert/X509Certificate;
    .locals 1

    .prologue
    .line 18
    const/4 v0, 0x0

    #v0=(Null);
    new-array v0, v0, [Ljava/security/cert/X509Certificate;

    #v0=(Reference);
    return-object v0
.end method

*/}
```

httpsurlbypass$2.java

```
package com.android; class httpsurlbypass$2 {/*

.class final Lcom/android/httpsurlbypass$2;
.super Ljava/lang/Object;
.source "httpsurlbypass.java"

# interfaces
.implements Ljavax/net/ssl/HostnameVerifier;


# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/android/httpsurlbypass;->hostname()Ljavax/net/ssl/HostnameVerifier;
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = null
.end annotation


# direct methods
```

```
.method constructor <init>()V
    .locals 0

    .prologue
    .line 34
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    .line 1
    #p0=(Reference);
    return-void
.end method


# virtual methods
.method public verify(Ljava/lang/String;Ljavax/net/ssl/SSLSession;)Z
    .locals 1
    .parameter "hostname"
    .parameter "session"

    .prologue
    .line 36
    const/4 v0, 0x1

    #v0=(One);
    return v0
.end method

*/}

bypass1.java
package com.android; class bypass1 {/*

.class public Lcom/android/bypass1;
.super Ljava/lang/Object;
.source "bypass1.java"


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 10
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    #p0=(Reference);
    return-void
.end method

.method public static byp()Landroid/webkit/WebViewClient;
    .locals 1

    .prologue
    .line 15
    new-instance v0, Lcom/android/bypass1$1;

    #v0=(UninitRef);
    invoke-direct {v0}, Lcom/android/bypass1$1;-><init>()V

    #v0=(Reference);
    return-object v0
.end method

.method public static webviewbypass(Landroid/webkit/WebView;)V
    .locals 1
    .parameter "view"

    .prologue
    .line 24
    invoke-static {}, Lcom/android/bypass1;->byp()Landroid/webkit/WebViewClient;

    move-result-object v0

    #v0=(Reference);
    invoke-virtual {p0, v0}, Landroid/webkit/WebView;->setWebViewClient(Landroid/
webkit/WebViewClient;)V

    .line 25
    return-void
.end method

*/}
```

bypass1$1.java

```
package com.android; class bypass1$1 {/*

.class final Lcom/android/bypass1$1;
.super Landroid/webkit/WebViewClient;
.source "bypass1.java"


# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/android/bypass1;->byp()Landroid/webkit/WebViewClient;
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = null
.end annotation


# direct methods
.method constructor <init>()V
    .locals 0

    .prologue
    .line 15
    invoke-direct {p0}, Landroid/webkit/WebViewClient;-><init>()V

    .line 1
    #p0=(Reference);
    return-void
.end method


 # virtual methods
.method public onReceivedSslError(Landroid/webkit/WebView;Landroid/webkit/
SslErrorHandler;Landroid/net/http/SslError;)V
    .locals 0
    .parameter “view”
    .parameter “handler”
    .parameter “error”

    .prologue
    .line 17
    invoke-virtual {p2}, Landroid/webkit/SslErrorHandler;->proceed()V

    .line 18
    return-void
.end method

*/}
```

References

- http://www.mcafee.com/us/resources/white-papers/foundstone/wp-pen-testing-android-apps.pdf
- http://developer.android.com/sdk/index.html
- http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html
- http://www.mcbsys.com/techblog/2010/12/android-certificates/
- http://code.google.com/p/android-apktool/
- http://blog.dahanne.net/2009/08/16/how-to-access-http-resources-from-android/
- http://www.netmite.com/android/mydroid/dalvik/docs/dalvik-bytecode.html
- http://www.love-android.net/RE-avance.pdf

About the Author

Naveen Rudrappa is a security consultant at McAfee Foundstone Professional Services, a division of McAfee. Naveen has more than four years of experience in information security. Naveen also hold CISSP, CEH, and SCJP certifications. At McAfee Foundstone, Naveen focuses on web application penetration testing, thick client testing, mobile application testing, web services testing, code review, threat modeling, external network penetration testing, and other service lines.

About McAfee Foundstone Professional Services

McAfee Foundstone Professional Services offers expert services and education to help organizations continuously and measurably protect their most important assets from the most critical threats. Through a strategic approach to security, McAfee Foundstone identifies and implements the right balance of technology, people, and process to manage digital risk and leverage security investments more effectively. The company's professional services team consists of recognized security experts and authors with broad security experience with multinational corporations, the public sector, and the US military.

About McAfee

McAfee, a wholly owned subsidiary of Intel Corporation (NASDAQ: INTC), empowers businesses, the public sector, and home users to safely experience the benefits of the Internet. The company delivers proactive and proven security solutions and services for systems, networks, and mobile devices around the world. With its visionary Security Connected strategy, innovative approach to hardware-enhanced security, and unique global threat intelligence network, McAfee is relentlessly focused on keeping its customers safe. http://www.mcafee.com.

McAfee
An Intel Company

2821 Mission College Boulevard
Santa Clara, CA 95054
888 847 8766
www.mcafee.com