使用Frida来hack安卓APP（二）-crackme

在第一篇介绍了Frida之后，我们现在打算来介绍如何使用Frida来解决简单的crackme。由于我们已经学习了关于Frida的一些知识，所以说，这一部分应该是简单的（理论上）。如果你想要跟着一起做一下的话，请下载

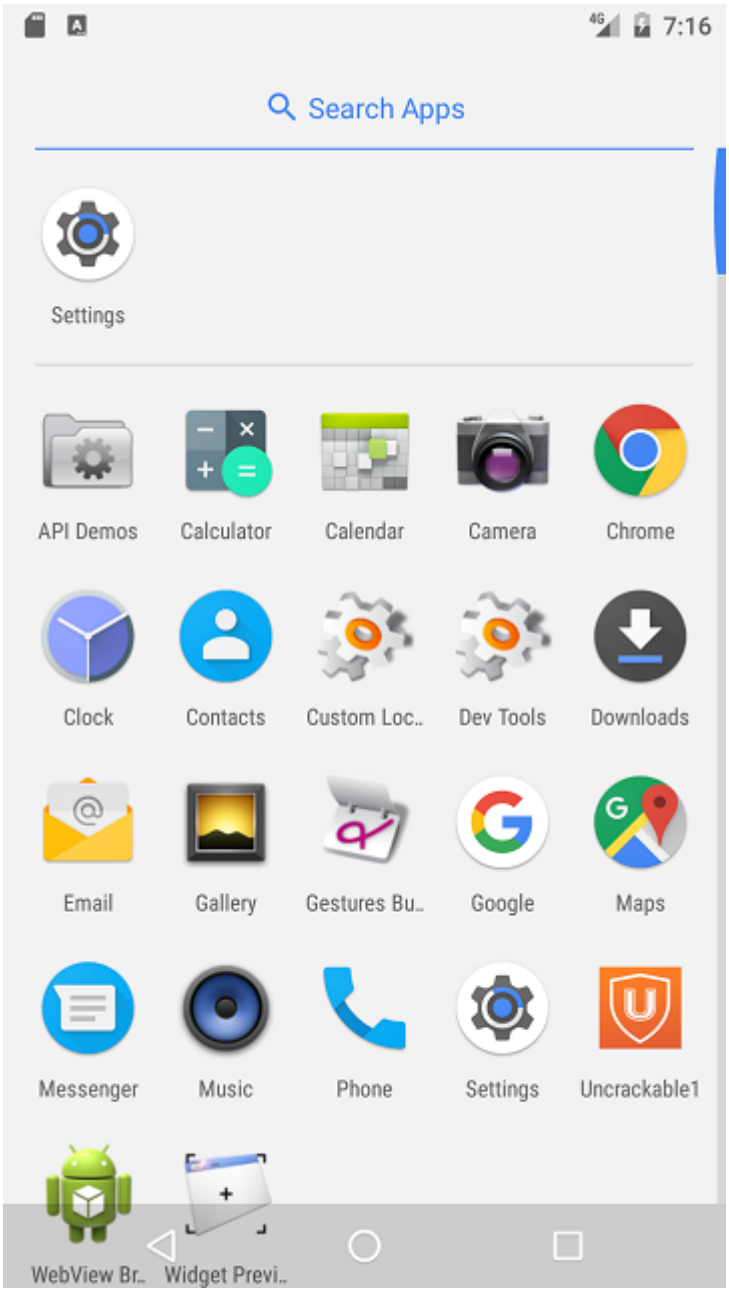
- the OWASP Uncrackable Crackme Level 1 (APK)
- BytecodeViewer
- dex2jar

同样，我也假设你已经成功安装了Frida(version 9.1.16 或者更新的版本)，同样在设备(rooted)上也启动了对应的server的binary。在这篇教程中，我打算使用Android 7.1.1 ARM镜像。

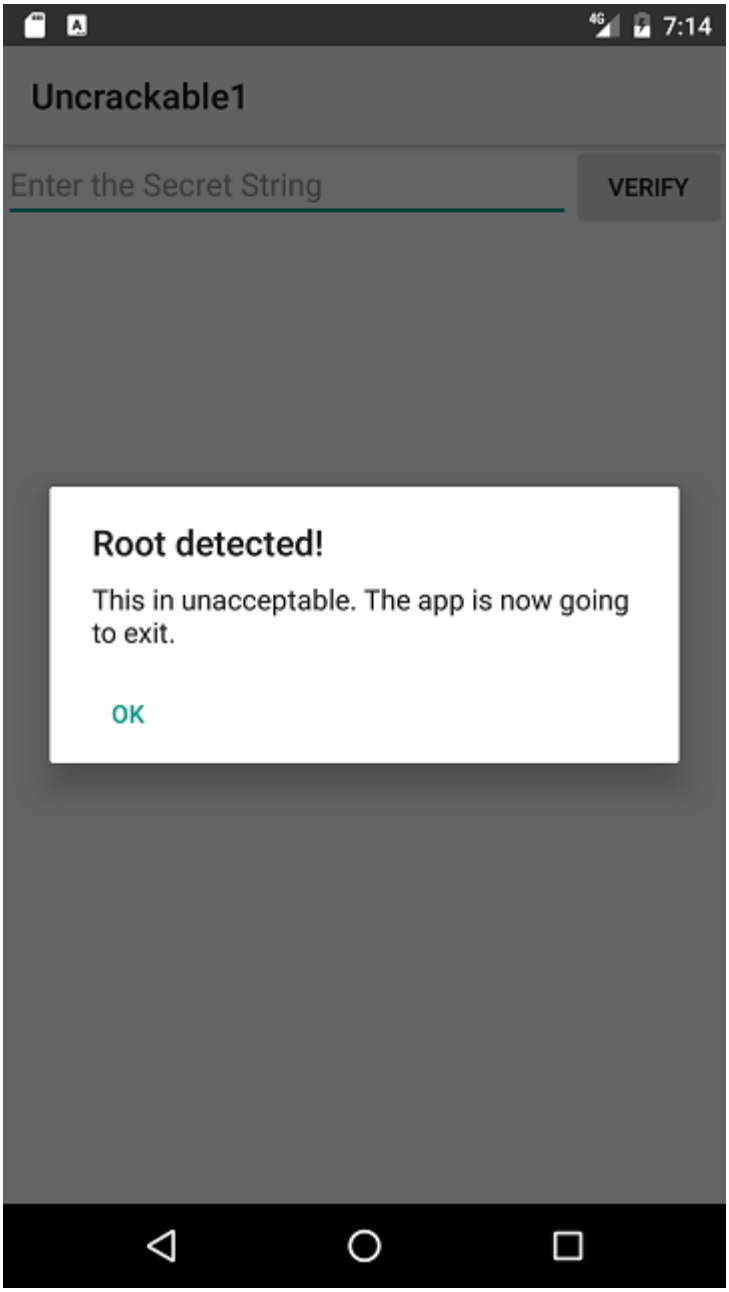
首先在你的设备上安装Uncrackable Crackme Level 1 app。

```
adb install sg.vantagepoint.uncrackable1.apk
```

等待他一直装完，然后从模拟器的菜单中打开它（右下角的橙色图标）：



当你打开那个app之后，你会发现，他并不想在一个已经rooted的设备上运行：



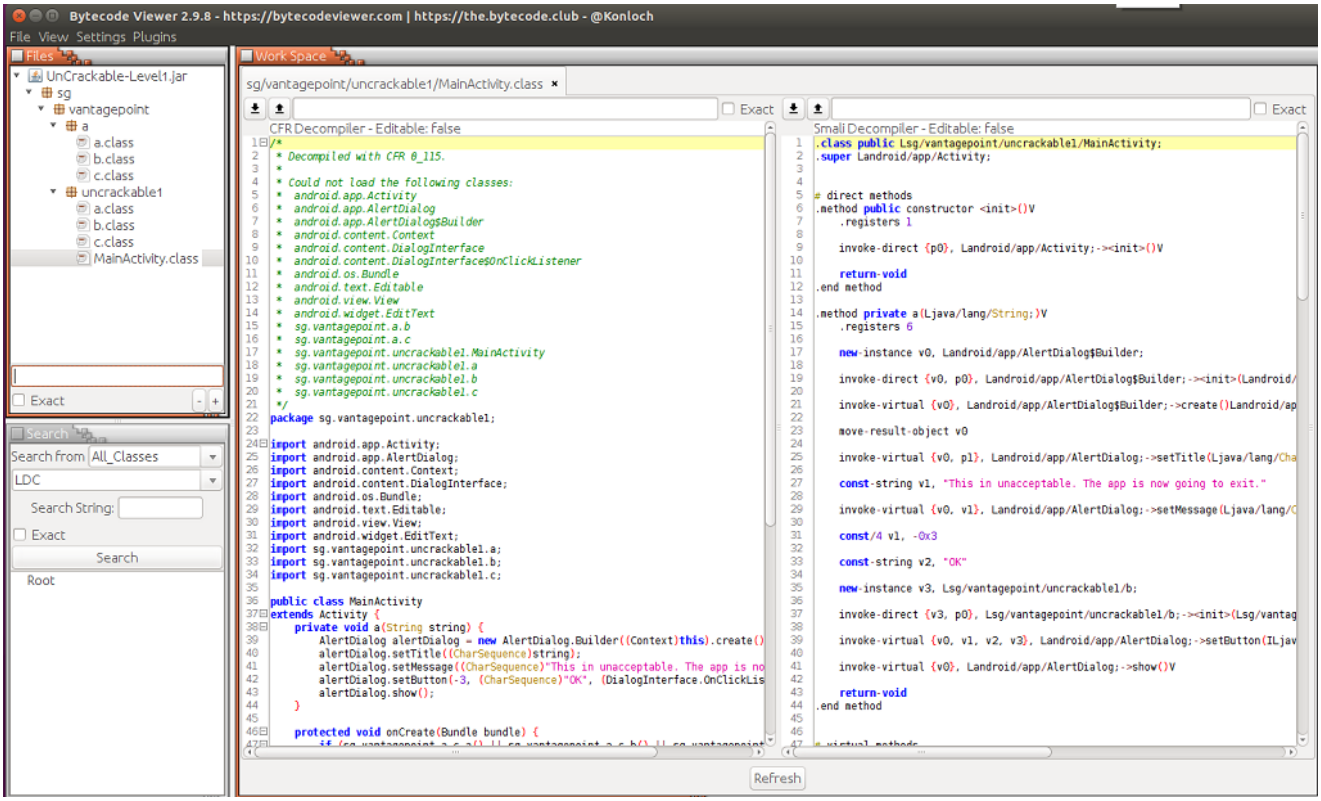
如果你点击OK，那个app就会立即退出。我们并不希望这样。看起来我们并不能这样去解决这个crackme。真的么？让我们来看一下到底发生了什么，以及app内部的一些细节。

我们可以利用dex2jar来讲一个apk转换为一个jar包。

```
michael@sixtyseven:/opt/dex2jar/dex2jar-2.0$ ./d2j-dex2jar.sh -o /home/michael/UnCrackable-
Level1.jar /home/michael/UnCrackable-Level1.apk

dex2jar /home/michael/UnCrackable-Level1.apk -> /home/michael/UnCrackable-Level1.jar
```

然后将它加载到BytecodeViewer（另一个支持java的反汇编器）中。你也可以直接将APK加载到BytecodeViewer或者仅仅是提取出来class.dex文件，但是我这样做之后并没有成功所以我利用dex2jar将其转换为了jar包。

在BytecodeViewer中，选择View->Pane1->CFR->Java来使用CFR反编译器。如果你想要比较反编译器的结果以及Smali代码，你可以设置Pane2为Smali，通常来说，这会比反编译结果更加准确一点。



这里是app的MainActivity被CFR反编译的结果。

```
package sg.vantagepoint.uncrackable1;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;
import android.text.Editable;
import android.view.View;
import android.widget.EditText;
import sg.vantagepoint.uncrackable1.a;
import sg.vantagepoint.uncrackable1.b;
import sg.vantagepoint.uncrackable1.c;

public class MainActivity
extends Activity {
    private void a(String string) {
        AlertDialog alertDialog = new AlertDialog.Builder((Context)this).create();
        alertDialog.setTitle((CharSequence)string);
        alertDialog.setMessage((CharSequence)"This in unacceptable. The app is now going to
exit.");
        alertDialog.setButton(-3, (CharSequence)"OK", (DialogInterface.OnClickListener)new
b(this));
        alertDialog.show();
    }

    protected void onCreate(Bundle bundle) {
        if (sg.vantagepoint.a.c.a() || sg.vantagepoint.a.c.b() || sg.vantagepoint.a.c.c()) {
            this.a("Root detected!"); //This is the message we are looking for
        }
        if (sg.vantagepoint.a.b.a((Context)this.getApplicationContext())) {
            this.a("App is debuggable!");
        }
        super.onCreate(bundle);
        this.setContentView(2130903040);
    }

    public void verify(View object) {
        object = ((EditText)this.findViewById(2131230720)).getText().toString();
        AlertDialog alertDialog = new AlertDialog.Builder((Context)this).create();
        if (a.a((String)object)) {
            alertDialog.setTitle((CharSequence)"Success!");
            alertDialog.setMessage((CharSequence)"This is the correct secret.");
        } else {
            alertDialog.setTitle((CharSequence)"Nope...");
            alertDialog.setMessage((CharSequence)"That's not it. Try again.");
        }
        alertDialog.setButton(-3, (CharSequence)"OK", (DialogInterface.OnClickListener)new
c(this));
        alertDialog.show();
    }
}
```

简单看一看其它被反编译的class文件，我们可以得知这是一个很小的app，我们甚至可以直接利用逆向工程解密和字符串修改的方法来解决这个问题。然而，由于我们知道Frida，我们有一些更为便捷的方法。让我们来看一看app在哪里检查了设备是否被rooted了。就在“Root detected”信息的上方，我们可以看到

```
if (sg.vantagepoint.a.c.a() || sg.vantagepoint.a.c.b() || sg.vantagepoint.a.c.c())
```

如果你简单看一看sg.vantagepoint.a.c，你就会看到不一样的对于root的检查。

```
public static boolean a()
    {
        String[] a = System.getenv("PATH").split(":");
        int i = a.length;
        int i0 = 0;
        while(true)
        {
            boolean b = false;
            if (i0 >= i)
            {
                b = false;
            }
            else
            {
                if (!new java.io.File(a[i0], "su").exists())
                {
                    i0 = i0 + 1;
                    continue;
                }
                b = true;
            }
            return b;
        }
    }

    public static boolean b()
    {
        String s = android.os.Build.TAGS;
        if (s != null && s.contains((CharSequence)(Object)"test-keys"))
        {
            return true;
        }
        return false;
    }

    public static boolean c()
    {
        String[] a = new String[7];
        a[0] = "/system/app/Superuser.apk";
        a[1] = "/system/xbin/daemonsu";
        a[2] = "/system/etc/init.d/99SuperSUDaemon";
        a[3] = "/system/bin/.ext/.su";
        a[4] = "/system/etc/.has_su_daemon";
        a[5] = "/system/etc/.installed_su_daemon";
        a[6] = "/dev/com.koushikdutta.superuser.daemon/";
        int i = a.length;
        int i0 = 0;
        while(i0 < i)
        {
            if (new java.io.File(a[i0]).exists())
            {
                return true;

            }
```

```
            i0 = i0 + 1;
        }
        return false;
    }
```

使用Frida，我们可以根据这个教程的第一部分来将所有这些函数都返回false。但是当它返回True的时候，到底发生了什么呢？真的是它检测到root了么？正如我们在`MainActivity`看到的，函数`a`打开了一个对话，同样它也设置了一个`onClickListener`，当我们按`ok`按钮的时候，这个函数就会被触发。

```
alertDialog.setButton(-3, (CharSequence)"OK", (DialogInterface.OnClickListener)new b(this));
```

这个`onClickListener`并没有做太多的事情

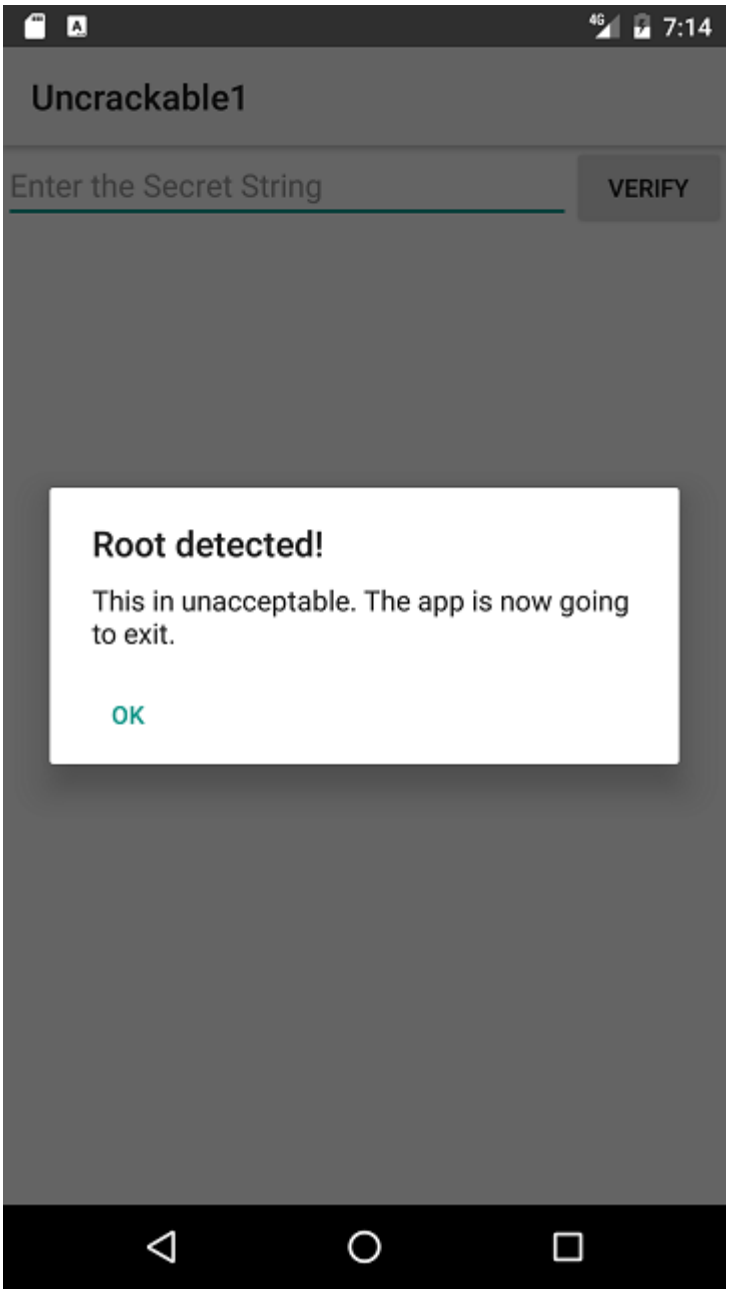
```
package sg.vantagepoint.uncrackable1;

class b implements android.content.DialogInterface$OnClickListener {
    final sg.vantagepoint.uncrackable1.MainActivity a;

    b(sg.vantagepoint.uncrackable1.MainActivity a0)
    {
        this.a = a0;
        super();
    }

    public void onClick(android.content.DialogInterface a0, int i)
    {
        System.exit(0);
    }
}
```

他就是简单地使用`system.exit(0)`来退出app。因此我们需要做的就是阻止App退出。我们来使用Frida把这个函数覆盖一下。创建`uncrackable.js`文件，如下

```
setImmediate(function() { //prevent timeout
    console.log("[*] Starting script");

    Java.perform(function() {

      bClass = Java.use("sg.vantagepoint.uncrackable1.b");
      bClass.onClick.implementation = function(v) {
         console.log("[*] onClick called");
      }
      console.log("[*] onClick handler modified")

    })
})
```

如果你看过这部分教程的 part I， 你应该能够很容易理解这一部分的代码。我们使用`setImmediate`来包装我们的函数以便于防止timeout(你可能不需要这个)。然后调用`Java.perform`来利用Frida与Java交互的函数。然后就会发生如下的事情：我们检索该类的wrapper,实现`OnClickListener`接口，并且覆盖`OnClick`方法。在我们的版本中，这个函数就只是在 命令行输出一些信息。同时，我们的app也不退出了。这正是因为我们替换了原有的

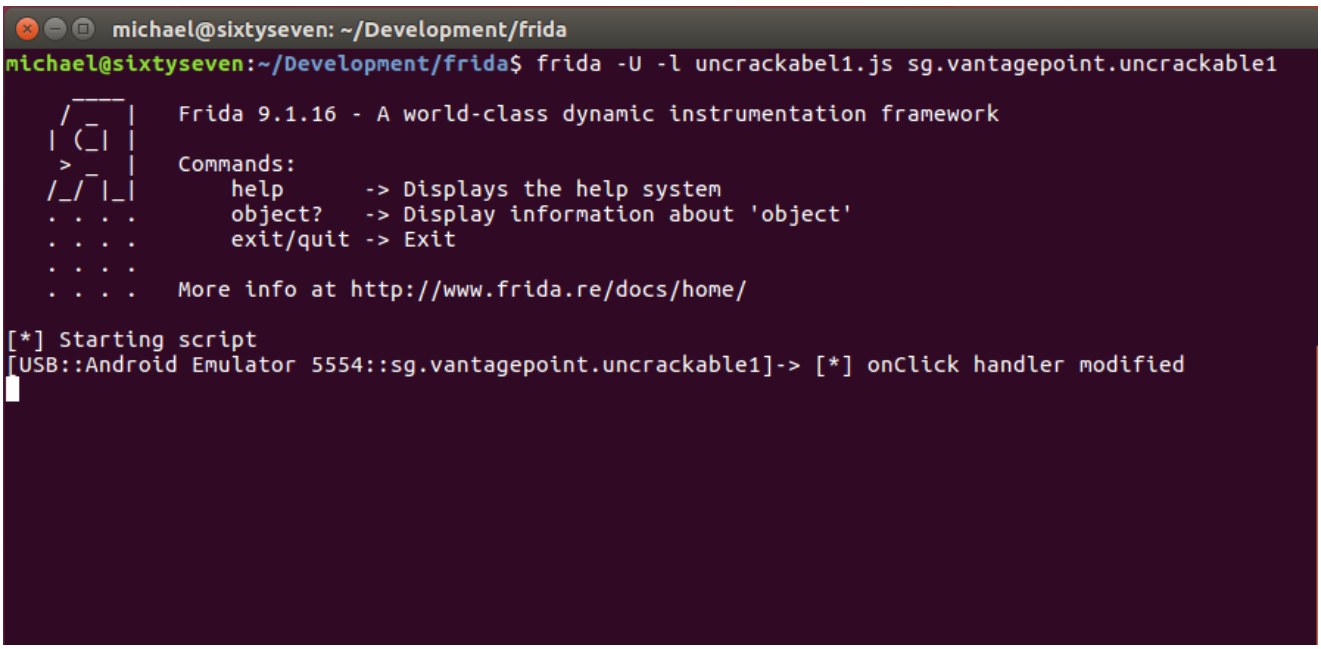
`OnClick`方法。让我们来试一下把，打开app，显示"Root detected"信息



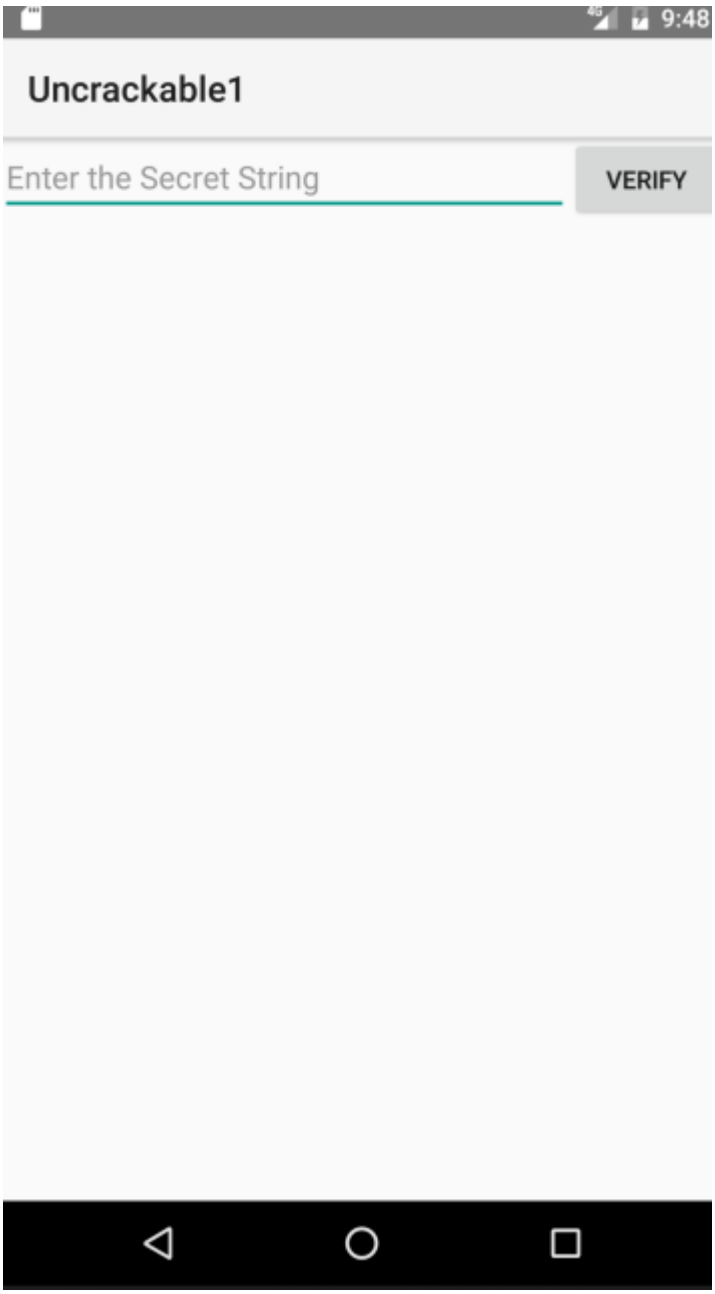
然后插入代码

```
frida -U -l uncrackable1.js sg.vantagepoint.uncrackable1
```

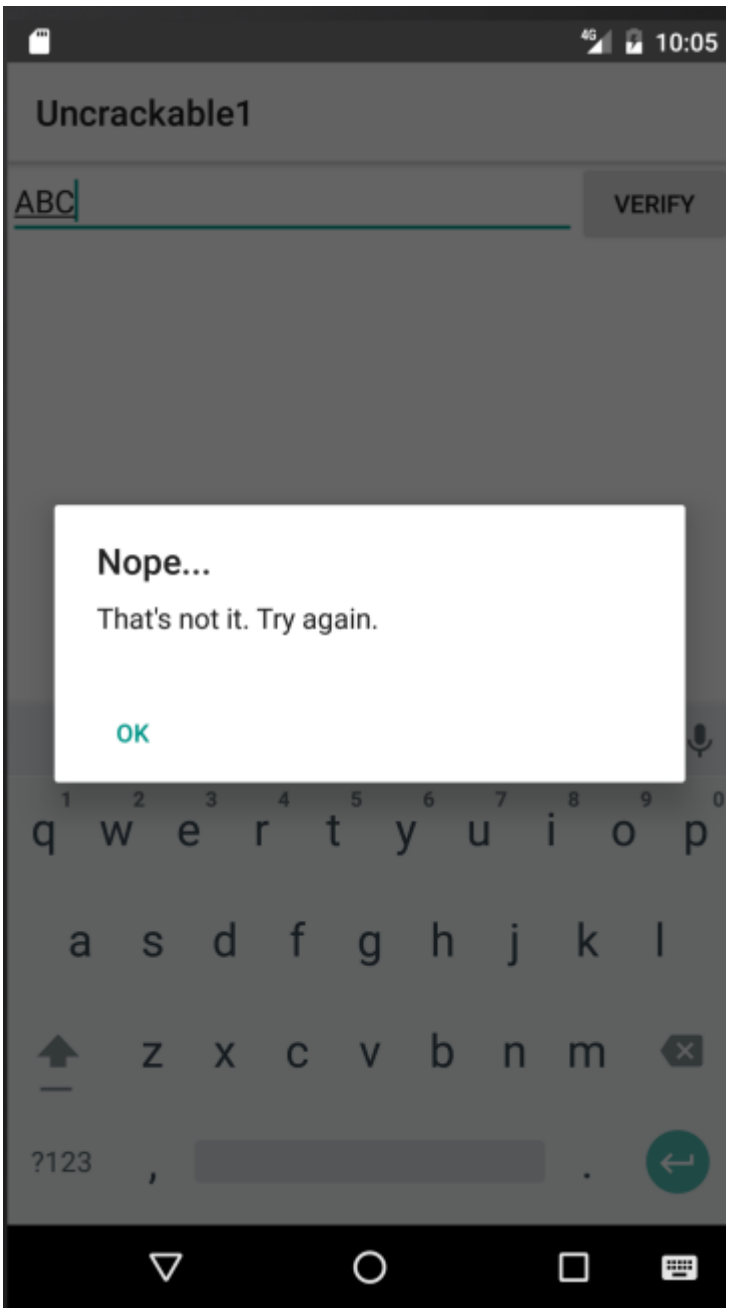
我们等待一会，当我们看到"onClick handler modified"的时候，说明 Frida已经成功插入了代码（你可能也会得到一个shell，因为我们将代码放在一个`setImmediate`的wrapper中，以便于Frida可以在后台执行）。



然后我们点击OK按钮，如果没什么问题的话，app就不会退出的。



棒极了，对话框消失了，现在我们可以去输入密码了。简单输入点东西，然后看一下发生了什么？



返回错误。但是显然，我们会有这样的一个想法，那就是应该是我们的输入会被加密，然后会和某个值比较，最后返回结果。

我们再来看一下`MainActivity`，我们看到了这样的函数

```
public void verify(View object) {
```

它调用了一个来自类`sg.vantagepoint.uncrackable1.a`的方法a。

```
if (a.a((String)object)) {
```

下面是类`sg.vantagepoint.uncrackable1.a`反编译后的结果。

```
package sg.vantagepoint.uncrackable1;

import android.util.Base64;
import android.util.Log;

/*
 * Exception performing whole class analysis ignored.
 */
public class a {
    public static boolean a(String string) {
        byte[] arrby = Base64.decode((String)"5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=",
(int)0);
        byte[] arrby2 = new byte[]{};
        try {
            arrby2 = arrby =
sg.vantagepoint.a.a.a((byte[])a.b((String)"8d127684cbc37c17616d806cf50473cc"), (byte[])arrby);
        }
        catch (Exception var2_2) {
            Log.d((String)"CodeCheck", (String)("AES error:" + var2_2.getMessage()));
        }
        if (!string.equals(new String(arrby2))) return false;
        return true;
    }

    public static byte[] b(String string) {
        int n = string.length();
        byte[] arrby = new byte[n / 2];
        int n2 = 0;
        while (n2 < n) {
            arrby[n2 / 2] = (byte)((Character.digit(string.charAt(n2), 16) << 4) +
Character.digit(string.charAt(n2 + 1), 16));
            n2 += 2;
        }
        return arrby;
    }
}
```

我们注意到方法a的最后面是将输入的字符串与经过处理的arrby2比较，判断是否相等。arrby2的值为`sg.vantagepoint.a.a.a`的返回结果。因此，我们下面需要考虑的就是该方法的返回值。

我们下面就可以开始对该函数进行逆向工程处理了。或者我们仅仅需要hook一下这个方法，然后让程序自己执行，然后返回对应的返回值即可。下面的脚本做的就是这个hook的事情。

```
        aaClass = Java.use("sg.vantagepoint.a.a");
        aaClass.a.implementation = function(arg1, arg2) {
            retval = this.a(arg1, arg2);
            password = ''
            for(i = 0; i < retval.length; i++) {
               password += String.fromCharCode(retval[i]);
            }

            console.log("[*] Decrypted: " + password);
            return retval;
        }
        console.log("[*] sg.vantagepoint.a.a.a modified");
```

我们覆盖了函数`sg.vantagepoint.a.a.a`，并且捕获了它的返回值并将其转换为一个可阅读的字符串。这也正是我们希望得到的结果。

我们把所有的部分合并在一下

```
setImmediate(function() {
    console.log("[*] Starting script");

    Java.perform(function() {

        bClass = Java.use("sg.vantagepoint.uncrackable1.b");
        bClass.onClick.implementation = function(v) {
         console.log("[*] onClick called.");
        }
        console.log("[*] onClick handler modified")


        aaClass = Java.use("sg.vantagepoint.a.a");
        aaClass.a.implementation = function(arg1, arg2) {
            retval = this.a(arg1, arg2);
            password = ''
            for(i = 0; i < retval.length; i++) {
               password += String.fromCharCode(retval[i]);
            }

            console.log("[*] Decrypted: " + password);
            return retval;
        }
        console.log("[*] sg.vantagepoint.a.a.a modified");


    });

});
```

先把它保存为`uncrackable1.js`，然后运行这个脚本（如果Frida不自动运行的话）

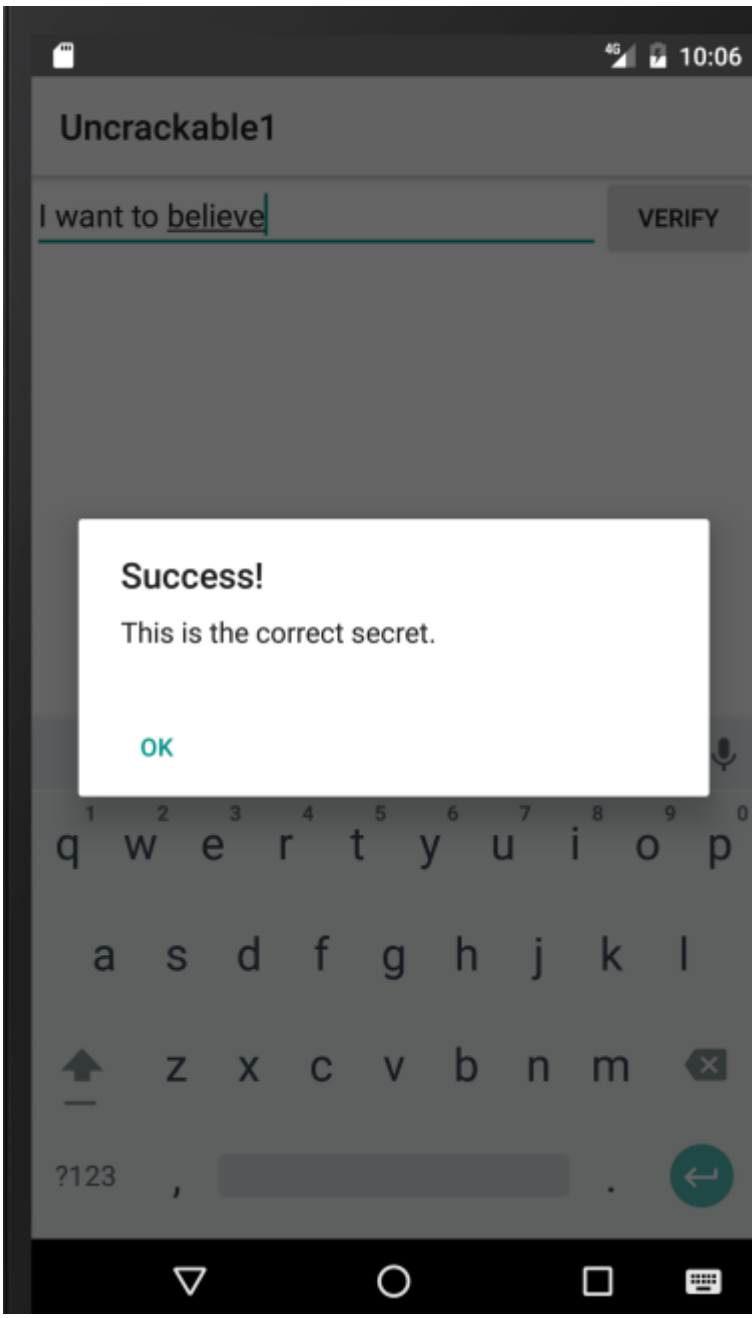
```
frida -U -l uncrackable1.js sg.vantagepoint.uncrackable1
```

直到你看到了消息`sg.vantagepoint.a.a.a modified`，然后点击`OK`，之后在security code处输入一点东西，并且按下Verify。

我们注意到了如下的结果

```
michael@sixtyseven:~/Development/frida$ frida -U -l uncrackable1.js sg.vantagepoint.uncrackable1
     ____
    / _  |   Frida 9.1.16 - A world-class dynamic instrumentation framework
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/

[*] Starting script
[USB::Android Emulator 5554::sg.vantagepoint.uncrackable1]-> [*] onClick handler modified
[*] sg.vantagepoint.a.a.a modified
[*] onClick called.
[*] Decrypted: I want to believe
```

漂亮，我们得到了对应的解密的字符串`I want to believe`。我们来检查一下是否正确



现在，我觉得你应该知道通过Frida可以做哪些事情了，以及它的动态插桩能力。

如上一篇所说的，如果有任何评论之类的，请在 Twitter 上联系我。

原文链接：https://www.codemetrix.net/hacking-android-apps-with-frida-2/

本文由看雪翻译小组 iromise 翻译。