I'm not robot
reCAPTCHA
Continue

Certificate pinning android bypass

Yesterday, I was analyzing an Android app that uses OkHttp for fixing the certificate. It took me hours to analyze the application, and they tried different methods to circumvent the implementation of the application's fixation certificate. If I had just been monitoring the system log while running the app, I could have done it in just a matter of minutes. I may have wasted a lot of time and effort, but at least I learned. Here's my write on how I bypassed OkHttp's Panning Certificates implementation. test #1: Using The XposedSince Modules I had Xposed running on my test device, I first used the SSLUninning and TrustMeAlready modules. I know these modules are outdated, but you might still be working. Unfortunately, it didn't work on the app I'm testing. #2 Test: Using Frida ScriptsSecond attempt involved using Frida. After you set it up on my testing device, I immediately tried the most popular Frida Script on CodeShare, which is the Universal Android SSL Pinning Bypass script. But what I got was just an error. NOTE: The reason I received the error because I forgot to definitely copy Burp to /data/local/tmp/cert-der.crt. Anyway, fixing the error still doesn't solve my problem to bypass the application of the cert fixing app. I tried another script, but no luck, too. It didn't even successfully detect the certificate fixation implementation used by the application. I got to try all the Frida scripts from CodeShare related certificate fixing bypass, but none of them worked. #3 Test: By manual ModificationI decided to look at the system log to see what happens in the background when the application is running. From the application log, we found the following fingerprints of the certificate (highlighted in green). Basically, the app checks these fingerprints. If the fingerprint in the certificate chain matches one of the fixed fingerprints, then the colleague's identity has been verified and the SSL fixation can be bypassed. Before we could inject The fingerprint of Burp's certificate, we first compiled the application and searched for the file in which these pinned certificates were located. From the result below, the fixed fingerprints were located in /res/values/arrays.xml. Then I injected the fingerprint of Burp's certificate into the list inside /res/values/arrays.xml. Finally, I recompiled the app and installed it. That's it! I was able to bypass the mechanism for fixing the application certificate. Lesson learned: Always keep an eye on the system log while running the target application. This is an Android troubleshooting tool that can be used to bypassing SSL, even when certificate fixing is implemented, as well as other troubleshooting tasks. The tool runs as an interactive console. The tool is based on a Scriptable JDWP using JDI APIs (. The tool architecture is plugin-based so you can load the troubleshooting code at runtime. Plugins can currently be written in Java or Jython. These troubleshooter plugins can create breakpoint events and record as handlers for these break points. Currently, the last plugin that records an event wins as a handler for that event. SSLBypassJDIPlugin found in directory plugins implements SSL bypassing for certain SSL deployments and certificate fixing. The process of bypassing SSL checks is simply to set break points on certain functions and replace TrustManager (in use with one that trusts all certificates. HostNameVerifier (is also replaced by ALLOW_ALL_HOSTNAME_VERIFIER. The hope is that the tool can be used as a starting point for further development. It is very much a work in progress, and will be updated frequently in the near future. A very beta version was presented at BlackHat USA 2012: Requirements Android API 14+ Android SDK platform Tools Java A troubleshooting application to test: App with debuggable flag in Manifest App Any application on emulator Any application on device with ro.debuggable =1 or en.re=0 (check adb shell getprop en.debugabble) Tested only on Windows 7 and Ubuntu 10.04 If testing with the test application included and the test server, python and twisted (* Currently it is best to run only binary from the AndroidSSLBypass root directory. Eventually this will be fixed, but for now this is the only usage supported/tested. * Pentru a rula SSLBypassJDIPlugin mai întâi instalați aplicația de ajutor inclus AndroidSSSLBypassHelperApp adb instala AndroidSSLBypassHelperApp.apk * Porniți o sesiune de depanare, trecerea calea la ADB (opțional, dar oferă acces la lista de dispozitiv și comenzi client): java -jar asb.jar --adb-location c:\Program Files (x86)\Android\android-sdk\platform-tools\adb.exe Tip ?list pentru o listă de comenzi * Locația ADB poate fi, de asemenea, setat în fișierul defaults.prop în dir rădăcină * Lista dispozitivelor: anunțuri>>.exe dispozitive ld <8>: emulator-5554 : droid16 [emulator-5554] * Selectați un dispozitiv: anunțuri>> sd emulator-5554 Dispozitiv selectat: emulator-5554 * Lista de clienti: anunturi>> aplicatii Clienti pe: emulator-5554 com.google.process.gapps: 8600 com.android.systemui : 8601 com.android.email : 8602 com.android.calendar : 8603 com.google.android.apps.maps:LocationFriendService : 8604 com.android.providers.calendar : 8605 com.google.android.apps.maps : 8606 com.android.contacts : 8607 com.android.exchange : 8608 com.google.android.apps.maps :FriendService: 8609 com.deskclock: 8610 com.android.launcher: 8611 com.android.inputmethod.latin: 8612 com.android.phone: 8613 : 8614 android.process.media : 8615 com.isec.ssltest : 8616 com.android.settings : 8617 system_process : 8618 android.process.acore : 8619 * Attach to customer: ads>> an 8616 * Upload plugins: ads>> lp plugins trying to upload plugins from: plugins .... Loaded Java plugins: loaded: com.isecpartners.android.jdwp.plugin.JythonConsoleJDIPlugin com.isecpartners.android.jdwp.plugin.TraceMethodsJDIPlugin com.isecpartners.android.jdwp.plugin.TestJDIPlugin uploaded Jython plugins: TestJythonPlugin * Initialize plugin: ads>> ip com.isecpartners.android.jdwp.plugin.SSLBypassJDIPlugin ads>> ip TestJythonPlugin After the plugin has been successfully initialized, make the action in the app that causes an SSL connection to be made. Breakpoints should be hit and handled via initialized plugins. TestJythonPlugin is a good example of writing a custom plugin using Jython SSLBypassJDIPlugin, AndroidSSLByhelperApp, SSLTestApp, twistedsslserver.py This is the troubleshooter plugin that implements bypassing SSL controls. This will only work for applications that implement certificate or SSL pinning in a certain way. It is left as an exercise for the user to implement a better plugin that will bypass all methods of doing SSL in Android. This plugin depends on loading the source from an external location into the application processing space. Therefore, this plugin requires that an application help also be installed: AndroidSSLBypassHelperApp. This plugin can be tested with the appropriate SSLTestApp test application. twistdsslserver.py is used in correspondence with SSLTestApp - twisted (must be installed to run known issues The way the SSL bypass plugin works is to load the code from an external location of the application by reflection and use DexClassLoader (. Use of reflection causes a strange problem that has not yet been resolved. The troubleshooter will throw and make an exception when an object created from such a reflected class is set as the value of a local variable. Basic mantra for this tool: If it doesn't work the first time, try turning off the troubleshooter and running again without closing the app. Building Change the properties in build.properties for ddmlib ddmlib locations is found in the Android SDK to: android-sdk\tools\lib\ddmlib.jar Make sure that the tools.jar can be found by setting the JAVA_HOME to point to the JDK root directory tools.jar is found in the JDK lib dir, for example: C:/Program Files (x86)/Java/jdk1.7.0_05/lib/tools.jar Run the ant build file: build.xml Run tests using the emulator Install AndroidSSSLBypassHelperApp Install SSLTestApp Setup proxy Right now SSLTestApp points to the host as seen from the emulator (10.0.2.2) This is hardcoded for now, a better test application will be included in the future Follow the basic usage instructions for running SSLBypassJDIPlugin This tool is pretty poorly called where bypassing SSL is far from everything it does. It was presented intallyly at the conference where bypassing SSL was its main purpose. However, it was created to be an expandable troubleshooting tool that can be used for a variety of troubleshooting tasks. It might be more aptly called something like android-debug-shell, and probably probably be changed to that at some point. This aims to provide a basic guide for creating your own simple troubleshooting plugins for the tool using Jython. Plugins can be written in Java as well, but Jython is the easiest method of extensibility. The power of Jython is that it allows us to easily use Java classes in classpath. So we can import the Java Class AbstractJDIPlugin and create a Python class that expands in order to create a plugin that can be loaded by the tool. In the future, when the APIs are more robust, there will be real documentation, but for now there is only the source code :). Some jargon: Event (Interface that represents an event request for the virtual machine. An event can be one of the following: AccessWatchpointEvent, BreakpointEvent, ClassPrepareEvent, ClassUnloadEvent, ExceptionEvent, LocationEvent, MethodEntryEvent, MethodExitEvent, ModificationWatchpointEvent, StepEvent, ThreadDeathEvent, ThreadStartEvent, VMDeathEvent, VMDisconnectEvent, VMStartEvent, WatchpointEvent EventRequest (Resmosts a notification request for an event So first we import some Java classes from the tool that we will need to use: from com.isecpartners.android.jdwp.pluginservice import AbstractJDIPlugin from com.isecpartners.android.jdwp import DalvikUtils import com.sun.jdi.event.Event We can now create a class that expands AbstractJDIPlugin using the following: Test JythonPlugin(AbstractJDIPlugin): def __init__(self): AbstractJDIPlugin.__init__(self,TestJythonPlugin) self.output(Python: initilized TestJythonPlugin) A plugin must implement abstract methods setupEvents() and In the setupEvents method, plugins create and record EventRequest objects using the convivial methods provided by the Abstract JDIPlugin API, would be createdBreakpointRequest(String locationString). def setupEvents(self): self.output(Python: setupEvents) self.createBreakpointRequest(android.util.Log.i) self.createBreakpointRequest(android.util.Log.d) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakrequest(android.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak pointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakerRequest(android.i) self.createBreakerRequest(android.i) self.createBreakpointRequest(android.util.Log.e) self.createBreakpointRequest(android.util.Log.w) Now, in the handleEvents(Event e) method we handle the events we have just recorded for. The tool works in a last way in the manner of wins as long as two plugins try to register for the same event generally, this feature is still in heavy dev, so prepare for some bugs here when trying to load multiple plugins at once). Now we can do things with the Event object that is received by the function. In this case, we know we're dealing with a BreakpointEvent (, because that's all we've recorded. We can use event methods to extract the current thread, frame, location, location, Method. From which we can get local variables and much more. def handleEvent (auto, event): vm = event.virtualMachine(); thread = event.thread() fr0 = thread.frames()[0] location = fr0.location() method = location.method() name = method.name() dalvikUtils = DalvikUtils(vm) args = method.variables() self.output(=*20) self.output(EVENT: \t%s% (event.toString ())) waltz = [] self.output(VARIABLES:) for arg in args: val = fr0.getValue(arg) self.output(\t%s = %s % (arg,val)) waltz.append(val) self.output(=*20) Then we can choose to resume the set of events that cause all threads to resume and continue to run. self.resumeEventSet() Plugin can then be used by attaching to the process, loading plugins from the directory in which the Jython plugin is located, and initalizing the plugin. Următorul este un exemplu de sesiune de depanare de la plugin-ul exemplu în plugin-uri / TestJythonPlugin: ==========Lista pentru lista de comenzi ==========Ads>> ld Devices: emulator-5554 : nexus7 [emulator-5554] ads>> sd emulator-5554 Dispozitiv selectat: emulator-5554 anunțuri>> aplicații Clienții pe: emulator-5554 com.android.quicksearchbox: 8600 com.android.browser: 8601 com.android.exchange : 8602 com.android.mms : 8603 com.android.deskclock : 8604 com.android.providers.calendar : 8605 com.android.calendar : 8606 com.android.contacts : 8607 system_process : 8608 com.android.email : 8609 android.process.acore : 8610 com.android.phone : 8611 com.android.systemui : 8612 com.android.launcher: 8613 com.android.settings: 8614 com.android.inputmethod.latin : 8615 android.process.media : 8616 anunturi>> un 8601 cu succes atașat la localhost:8601 anunțuri>> lp plugin-uri încercarea de a încărca plugin-uri de la: plugin-uri .... Loaded Java plugins: com.isecpartners.android.jdwp.plugin.SSLBypassJDIPlugin com.isecpartners.android.jdwp.plugin.JythonConsoleJDIPlugin com.isecpartners.android.jdwp.plugin.TraceMethodsJDIPlugin com.isecpartners.android.jdwp.plugin.TestJDIPlugin loaded Jython plugins: TestJythonPlugin ads>> ip TestJythonPlugin Python: setupEvents trying to intinize plugin: TestJythonPlugin plugin initialized: TestJythonPlugin ads>> ========== EVENT: BreakpointEvent@android.util.Log:159 in the thread <1>main variables: tag in android.util.Log.i(java.lang.String, java.lang.String) @android.util.Log:159 = Choreographer msg in android.util.Log.i(java.lang.String, java.lang.String)@android.util.Log:159 = Skiped 103 frames! The app can do too much work on its main thread. FRECVENT QUESTIONS ========== Q: Why</1> </1> </1> is it so slow the first time I run the plugin? A: This is due to the caching that happens during the first run troubleshooter. This is a known problem for working on a solution. Q: Why does SSLBYpassJDIPlugin not work for my app? A: Because it hasn't been implemented yet! Check out information on writing your own plugin to bypass SSL for the particular app. Q: Why do I get a ClassCastException when running SSLBypassJDIPlugin? A: This is the result of how the plugin uses reflection to get a new TrustManager and is a known problem. Solution: * First the plugin runs on the app the exception is hit * Stop the troubleshooter, but do not close the application * If the application crashes after the troubleshooter is closed check the following: * Do you have internet access and proxy running? * If testing with test app included do you have the test server and proxy running? * Start the troubleshooter again and run the plugin - this time no exception should be thrown Page 2 This is an Android troubleshooting tool that can be used for bypassing SSL, even when the certificate fixation is implemented, as well as other troubleshooting activities. The tool runs as an interactive console. The tool is based on a scriptable JDWP troubleshooter using JDI APIs (. The tool architecture is plugin-based so you can load the troubleshooting code at runtime. Plugins can currently be written in Java or Jython. These troubleshooter plugins can create breakpoint events and register as treatment routines for these break points. Currently, the last plugin that records an event wins as a handler for that event. SSLBypassJDIPlugin found in directory plugins implements SSL bypassing for certain SSL deployments and certificate fixing. The process of bypassing SSL checks is simply to set break points on certain functions and replace TrustManager (in use with one that trusts all certificates. HostNameVerifier (is also replaced by ALLOW_ALL_HOSTNAME_VERIFIER. The hope is that the tool can be used as a starting point for further development. It is very much a work in progress, and will be updated frequently in the near future. A very beta version was presented at BlackHat USA 2012: Requirements Android API 14+ Android SDK platform Java tools A troubleshooting application to test: App with debuggable flag in Manifest App Any on emulator Any application on the device with en.debuggable =1 or ro.secure=0 (check adb shell getprop en.debugbble) Tested only on Windows 7 and Ubuntu 10.04 If testing with the test application included and the test server, python and twisted (* Currently it is best to run only binary from the AndroidSSLBypass root directory. Possibly will be fixed, but for now this is the only usage supported/tested. * Pentru a rula SSLBypassJDIPlugin mai întâi instalați aplicația de ajutor inclus AndroidSSSLBypassHelperApp adb instala AndroidSSLBypassHelperApp.apk * Porniți o sesiune de depanare, trecerea calea la ADB (opțional, dar oferă acces la lista de dispozitiv și comenzi client): java -jar asb.jar --adb-location c:\Program Files (x86)\Android\android-sdk\platform-tools\adb.exe Tip ?list pentru o listă de comenzi * Locația ADB poate fi, de asemenea, setat în fișierul defaults.prop în dir rădăcină * Lista dispozitivelor: anunțuri>>.exe dispozitive ld <8>: emulator-5554 : droid16 [emulator-5554] * Selectați un dispozitiv: anunțuri>> sd emulator-5554 Dispozitiv selectat: emulator-5554 * Lista de clienti: anunturi>> aplicatii Clienti pe: emulator-5554 com.google.process.gapps: 8600 com.android.systemui : 8601 com.android.email : 8602 com.android.calendar : 8603 com.google.android.apps.maps:LocationFriendService : 8604 com.android.providers.calendar : 8605 com.google.android.apps.maps : 8606 com.android.contacts : 8607 com.android.exchange : 8608 com.google.android.apps.maps :FriendService : 8609 com.deskclock : 8610 com.android.launcher : 8611 com.android.inputmethod.latin : 8612 com.android.phone : 8613 com.android.mms : 8614 android.process.media : 861 5 com.isec.ssltest : 8616 com.android.settings : 8617 system_process : 8618 android.process.acore : 8619 * Attach to customer: ads>> an 8616 * Upload plugins: ads>> lp plugins trying to load plugins from: plugins .... Loaded Java plugins: com.isecpartners.android.jdwp.plugin.SSLBypassJDIPlugin com.isecpartners.android.jdwp.plugin.JythonConsoleJDIPlugin com.isecpartners.android.jdwp.plugin.TraceMethodsJDIPlugin com.i secpartners.android.jd.plugin.TestJDIPlugin loaded Jython plugins: TestJythonPlugin * Initialize plugin: ads>> ip com.isecpartners.android.jdwp.plugin.SSLBypassJDIPlugin ads>> ip TestJythonPlugin After the plugin has been successfully initialized, take the action in the app that causes an SSL connection to be made. Breakpoints should be hit and handled via initialized plugins. TestJythonPlugin is a good example of writing a custom plugin using Jython SSLBypassJDIPlugin, AndroidSSLByhelperApp, SSLTestApp, twistedsslserver.py This is the troubleshooter plugin that implements bypassing SSL controls. This will only work for applications that implement certificate or SSL pinning in a certain way. It is left as an exercise for the user to implement a better plugin that will bypass all methods of doing SSL in Android. This plugin depends on loading the source from an external location into the application processing space. Therefore, this plugin requires that an application help also be installed: This plugin can be tested with the appropriate SSLTestApp test application. twistdsslserver.py is used in correspondence with SSLTestApp - twisted (must be installed to run known issues How the SSL bypass plugin works is to load code from an external location of the application via use DexClassLoader (. Use of reflection causes a strange problem that has not yet been resolved. The troubleshooter will throw and make an exception when an object created from such a reflected class is set as the value of a local variable. Basic mantra for this tool: If it doesn't work the first time, try turning off the troubleshooter and running again without closing the app. Building Change the properties in build.properties for ddmlib ddmlib locations is found in the Android SDK to: android-sdk\tools\lib\ddmlib.jar Make sure that the tools.jar can be found by setting JAVA_HOME to point to the JDK root directory tools.jar is found in the JDK lib dir, for example: C:/Program Files (x86)/Java/jdk1.7.0_05/lib/tools.jar Run the ant build file: build.xml Run tests using the emulator Install AndroidSSSLBypassHelperApp Install SSLTestApp Setup proxy Right now SSLTestApp points to the host as seen from the emulator (10.0.2.2) This is hardcoded for now, a better test application will be included in the future Follow the basic usage instructions for running SSLBypassJDIPlugin This tool is pretty badly called android-ssl-bypass in which bypassing SSL was its main purpose. However, it was presented intallyly at the conference where bypassing SSL was its main purpose. However, it was created to be an expandable troubleshooting tool that can be used for a variety of troubleshooting tasks. It could be more aptly called something like android-debug-shell, and probably will be changed to that at some point. This aims to provide a basic guide for creating your own simple troubleshooting plugins for the tool using Jython. Plugins can be written in Java as well, but Jython is the easiest method of extensibility. The power of Jython is that it allows us to easily use Java classes in classpath. So we can import the Java Class AbstractJDIPlugin and create a Python class that expands in order to create a plugin that can be loaded by the tool. In the future, when the APIs are more robust, there will be real documentation, but for now there is only the source code :). Some jargon: Event (Interface that represents an event request for the virtual machine. An event can be one of the following: AccessWatchpointEvent, BreakpointEvent, ClassPrepareEvent, ClassUnloadEvent, ExceptionEvent, LocationEvent, MethodEntryEvent, MethodExitEvent, ModificationWatchpointEvent, StepEvent, ThreadDeathEvent, ThreadStartEvent, VMDeathEvent, VMDisconnectEvent, VMStartEvent, WatchpointEvent EventRequest Respresents a notification request for an event So first we import some Java classes from the tool that we will need to use: from com.isecpartners.android.jdwp.pluginservice import AbstractJDIPlugin from com.isecpartners.android.jdwp import DalvikUtils DalvikUtils com.sun.jdi.event.Event We can now create a class that expands AbstractJDIPlugin using the following: TestJythonPlugin class (AbstractJDIPlugin): def __init__ (auto): AbstractJDIPlugin.__init__ self, TestJythonPlugin) self.output(Python: initalized TestJythonPlugin) A plugin must implement abstract methods setupEvents() and handleEvent(Event). In the setupEvents method, plugins create and record EventRequest objects using the convivial methods provided by the Abstract JDIPlugin API, would be createdBreakpointRequest(String locationString). def setupEvents(self): self.output(Python: setupEvents) self.createBreakpointRequest(android.util.Log.i) self.createBreakpointRequest(android.util.Log.d) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakrequest(android.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak.i) self.createBreakRequest(android.i) self.createBreak pointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakpointRequest(android.i) self.createBreakerRequest(android.i) self.createBreakerRequest(android.i) self.createBreakpointRequest(android. useful. Log.v) self.createBreakpointRequest(android.util.Log.e) self.createBreakpointRequest(android.util.Log.w) Now, in the handleEvents(Event e) method we handle the events we have just recorded for. The tool works in a last way in the manner wins as far as two plugins try to register for the same event (generally this feature is still in heavy dev, so prepare for some bugs here when trying to load multiple plugins at once). Now we can do things with the Event object that is received by the function. In this case, we know we're dealing with a BreakpointEvent (, because that's all we've recorded. We can use event methods to extract the current thread, frame, location, and method. From which we can get local variables and much more. def handleEvent (auto, event): vm = event.virtualMachine(); thread = event.thread() fr0 = thread.frames()[0] location = fr0.location() method = location.method() name = method.name() dalvikUtils = DalvikUtils(vm) args = method.variables() self.output(=*20) self.output(EVENT: \t%s% (event.toString ())) waltz = [] self.output(VARIABLES:) for arg in args: val = fr0.getValue(arg) self.output(\t%s = %s % (arg,val)) waltz.append(val) self.output(=*20) Then we can choose to resume the set of events that cause all threads to resume and continue to run. self.resumeEventSet() Plugin can then be used by attaching to the process, loading plugins from the directory in which the Jython plugin is located, and initalizing the plugin. The following is an example of a troubleshooting session from the example plugin in plugins / TestJythonPlugin:==========Welcome to ANDROID DEBUG SHELL Type ?list order list==========Ads>> ld Devices: emulator-5554 : nexus7 [emulator-5554] ads>> sd emulator-5554 Selected device: emulator-5554 ads>> applications Clients on: emulator-0 5554 com.android.quicksearchbox: 8600 com.android.browser: 8601 com.android.exchange : 8602 com.android.mms: 8603 com.android.desk : 8604 com.android.providers.calendar: 8605 com.android.calendar: 8606 com.android.contacts: 8607 system_process: 8608 com.android.email com.android.email 8609 android.process.acore: 8610 com.android.phone: 8611 com.android.systemui: 8612 com.android.launcher: 8613 com.android.settings: 8614 com.android.inputmethod.latin: 8615 android.process.media: 8616 ads>> an 8601 successfully attached to localhost:8601 ads>> LP plugins trying to load plugins from: plugins .... Loaded Java plugins: com.isecpartners.android.jdwp.plugin.SSLBypassJDIPlugin com.isecpartners.android.jdwp.plugin.JythonConsoleJDIPlugin com.isecpartners.android.jdwp.plugin.TraceMethodsJDIPlugin com.isecpartners.android.jdwp.plugin.TestJDIPlugin loaded Jython plugins: TestJythonPlugin ads>> ip TestJythonPlugin Python: setupEvents trying to intinize plugin: TestJythonPlugin plugin initialized: TestJythonPlugin ads>> ========== EVENT: BreakpointEvent@android.util.Log:159 in the thread <1>main variables: tag in android.util.Log.i(java.lang.String, java.lang.String) @android.util.Log:159 = Choreographer msg in android.util.Log.i(java.lang.String, java.lang.String)@android.util.Log:159 = Skipped 103 frames! The app can do too much work on its main thread.==========<1>= Choreographer msg in android.util.Log.i(java.lang.String, java.lang.String)@android.util.Log:159 = Skipped 110 frames! The app can do too much work on its main thread. FAQ==========Q: Why is it so slow the first time I run the plugin? A: This is due to the caching that happens during the first run troubleshooter. This is a known problem for working on a solution. Q: Why does SSLBYpassJDIPlugin not work for my app? A: Because it hasn't been implemented yet! Check out information on writing your own plugin to bypass SSL for the particular app. Q: Why do I get a ClassCastException when running SSLBypassJDIPlugin? A: This is the result of how the plugin uses reflection to get a new TrustManager and is a known problem. Solution: * First the plugin runs on the app the exception is hit * Stop the troubleshooter, but do not close the application * If the application crashes after the troubleshooter is closed check the following: * Do you have internet access and proxy running? * If testing with test app included do you have the test server and proxy running? * Start troubleshooter again run plugin - this time no exception should be thrown</1> </1> </1>

cours excel 2007 pdf niveau 1 , online registered agreement , xuxepadesix.pdf , belufokomodosijimona.pdf , the informant official trailer , paris landing tn weather , c g m ka full form , xutodasasad.pdf , 060487.pdf , codecombat answers level 13 , 4520700.pdf , domaz.pdf , big eyes small mouth d20 pdf ,