# “Debuggable” Flag - Android Application

Manifest file is the configuration file for android applications. Common attributes of manifest file are -

- **manifest** - Set version, package name, version name and high level information
- **uses-sdk** - Define the minimum and maximum SDK versions supported
- **uses-configuration** - Specify what is the type of input supported by the application i.e. trackpad, i-ball, keypad and so on
- **uses-feature** - Specify hardware requirement of the application
- **supports-screens** - Allows to specify which screen size application supports
- **application** - Most important part of the manifest file. Any manifest will have only one "application" tag. Application tag includes tags to specify icons, theme, application name, activity, services, provider and receiver.
- **user-permission** - List of permission requires to run application properly

One of the key attribute is "Debuggable" which describes whether the application is in debug mode or not. It is part of the "application" section in the manifest file.

On the device itself the “adbd” daemon is started as soon as the phone is connected to the computer when USB debugging is enabled. If “Debuggable”attribute is set o true, the application will try to connect to a local unix socket “**@jdwp-control**”. This socket is opened on the device by adbd, waiting for debuggable applications to register by connecting to the socket. If adbd is running it will accept the connections from debuggable applications and keep a list of the applications. Adb running on the computer can now request a connection to any of these and adbd will forward a connection between the debugger and the debuggee. Any further communication will now take place using the standard Java Debug Wire Protocol.

Using JDWP, It is possible to gain full access to the Java process and execute arbitrary code in the context of the debugable application.

CheckDebuggable is a ruby script which accepts APK file as input to find out whether debuggable flag is set to true in the given APK. It uses “apktool” to extract file from the APK. Here is a script –

```
filename=ARGV[1]

def h(filename)

case ARGV[0]

   when "-f"

     @filename=ARGV[1]
```

```
   else
     system('cls')
     puts "CheckDebug.rb -f <APPLNAME.apk> \n Example: CheckDebug.rb -f DumpDroid.apk"
     exit
  end
   n=filename.length


   cmp=filename[n-4,4]


 if cmp!='.apk'
 puts "enter file name with extension .apk \nfor eg CheckDebug.rb -f <APPLNAME.apk> \n Example:
CheckDebug.rb -f DumpDroid.apk "
 exit
 end
  if !File.exists?(@filename)
    puts "\nCannot find #{@filename}! Please make sure the path of the file is correct."
    Process.exit
  end
require 'rexml/document'


            name=filename
            folder=name.split(".apk")
            foldername=folder[0]


quot='"'
```

```
	cmd=Dir.getwd+"/apktool d #{quot}#{name}#{quot} ./#{quot}#{foldername}#{quot}"



	system(cmd)

	begin

	readxml = REXML::Document.new File.new("#{foldername}/AndroidManifest.xml")



		rescue

		puts 'check whether apktool batch file and apktool jar file are present in installation
folder'

		exit

	end

	check  = readxml.elements.to_a("//application")

	e = check[0]



	puts "  \nCheckDebugFlag v1.0 (beta) (c) 2012 Copyright and  All Rights Reserved. eSphere
Security \n http://www.espheresecurity.com \n Contact us at - contact@espheresecurity.net \n "

begin

	bflag= e.attribute("android:debuggable").value



	if bflag == 'true'

		puts 'android debuggable flag is on'

	else

		puts 'android debuggable flag is off'



	end
```

```
  rescue NoMethodError

                        puts 'android debuggable flag is off'

end



require 'fileUtils'

del=Dir.getwd+"/#{foldername}"

FileUtils.remove_dir("#{del}")

end

h (filename)
```

It is important to have apktool installed on the machine or extract the zip file which contains the required files (including apktool)