ARL-TN-0681 ● Oct 2015

US Army Research Laboratory

Porting Extremely Lightweight Intrusion Detection (ELIDe) to Android

by Ken F Yu and Garret S Payer

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0681 ● OCT 2015

US Army Research Laboratory

Porting Extremely Lightweight Intrusion Detection (ELIDe) to Android

by Ken F Yu and Garret S Payer
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
Oct 2015	Final	April 2014–September 2014

4. TITLE AND SUBTITLE
Porting Extremely Lightweight Intrusion Detection (ELIDe) to Android

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Ken F Yu and Garret S Payer

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIN-D
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0681

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report describes how to port Extremely Lightweight Intrusion Detection (ELIDe) from Linux to Android. A step-by-step guide is provided for engineers who want to develop ELIDe software for the Android operating system.

15. SUBJECT TERMS
ELIDe, Android, pcap

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ken F Yu
Unclassified	Unclassified	Unclassified	UU	32	19b. TELEPHONE NUMBER (Include area code) (301) 394-3181

Contents

1. Background

Current breakthroughs in mobile computing technology will allow unprecedented computational power and information from almost anywhere via hand-held devices. The explosion of mobile devices presents unique security concerns that may not have been considered because productivity software has migrated from a traditional wired-network architecture utilizing personal computers (PCs) to mobile phones entirely dependent on wireless communications.

Initial work using the Extremely Lightweight Intrusion Detection (ELIDe) system focused on replicating the malicious-packet detection that is achieved using larger, more computationally heavy systems through machine-learning techniques. If a combination of N-gram hashing, stochastic gradient descent, and linear classifiers is used, it is possible to closely match the performance of traditional packet classification but significantly reduce the computational load, which leads to lower power requirements.

Because ELIDe was found to be effective in performing network packet classification, it is now necessary to determine how this capability will perform on a mobile device such as a consumer-targeted Android phone. Ritchey et al.[1] established how to build the dependencies needed to run ELIDe on a mobile platform. This process included compiling both Basic Linear Algebra Subprograms (BLAS) and libpcap for the Android platform. The work described in this document continues the effort required to migrate ELIDe to Android.

2. Configuration

The following is a list of the software and hardware used as part of the development effort needed to migrate ELIDe to the Android platform.

2.1 Development Work Station

The work station has the following configuration:

- Operating System: Red Hat Enterprise Linux, version 6.5
- Android Development Tools, version 22.3.0-887826
- Android NDK Revision 9
- Dell Precision T7400
    - 8-GB memory

- Dual Xeon X5472 central processing unit
  - Quad Core
  - 3.00 GHz

2.2 Development Mobile Device

The following mobile devices were used to develop the mobile version of ELIDe:

- Android 4.3 Jelly Bean (Application Programming Interface Level 18)
- Sprint-branded Galaxy S3
  - Qualcomm Snapdraggon S4 Plus MSM8960
  - Dual-core 1.5-GHz Krait processor
  - Adreno 225 graphics processor
  - 2048 MB of RAM
  - 32-GB internal storage
  - 2100-mAh battery

3. Developing the Software

Much of this work extends from research efforts accomplished in Compilation of a Network Security/Machine Learning Toolchain for Android ARM Platforms.[1] However, it utilizes several other pieces of work that will be referenced throughout the document. The following steps were taken to port ELIDe to the Android platform.

- Root the Android device to use the libpcap to access the device's hardware interfaces
- Set up Android x86 on VirtualBox
- Build a new Android application
- Compile ELIDe's C++ and FORTRAN code for Android
- Generate the ELIDe native code at back end
- Develop the Android application at front end
- Deploy the application

Although there are several different ways to port the ELIDe functionality over to the Android platform, a C++ based native code for ELIDe was chosen. The Android ELIDe native code was developed based on the existing ELIDe Linux version with some modifications, which improved speed and performance as well as the network data-capturing capability. In addition, the same native code can be used to operate the software on both the Linux and Android operating systems. An advantage to using the non-native code is the Android device does not have to be rooted; however, the absence of the native code would mean dramatically decreased speed performance, and this makes the ELIDe application inoperable.

3.1 Rooting Android Device

"Rooting" refers to the process of gaining access to the administrative commands and functions of an operating system (OS). An Android device comes with a number of protections so that applications developed by the platform are not supposed to be able to run with a root context. However, in order to listen to the network communications transmitted and received on the wireless adapter, root privileges are required. Because Android does not provide root capabilities by default, it became necessary to take steps to root the device so that root privileges for specific ELIDe processes can be used.

The procedure needed to root an Android phone varies by device. The technical note Rooting Android Device[2] details how the experimental Android device is rooted. When virtualized Android x86 is used, the root context is allowed by default.

3.2 Android OS on Virtual Machine

A lot of the initial development for virtual machines was achieved without running code on the mobile device. The Android Software Development Kit (SDK) includes the ability to emulate several different Android devices, called Android Virtual Devices (AVDs). These AVDs are tools used to debug the Android software without using the actual Android devices. The major drawback to using AVDs is that they perform a direct translation for ARM instructions to x86 instructions in order to run Android applications on work stations equipped with an Intel or Advanced Micro Devices processor. Although this process allows for a more accurate reproduction of the application, AVDs run much slower than actual hardware because of the translation. The slowdown may intensify significantly if the application developed is intended for a heavy computational load. Due to the slow performance of AVDs, the application cannot handle relatively high-speed network traffic during testing. In order to test the application on the development

work station, a version of Android for x86 processors was utilized. This technique can then be virtualized on the development work station and integrated into the compilation process used by the Android SDK. Although this representation is not as accurate as the application running on a device, it was more than satisfactory for study purposes. The report Android Virtual Machine (VM) Setup on Linux[3] has a more detailed explanation of this method.

3.3 Building a New Android Application

The ELIDe application is built based on the Default Android Auto-Generated Android project. To build a new Android application, follow these steps:

1. Use Eclipse from the Android SDK. From File->New->Android Application Project, it will bring up the New Android Project setting page.
2. Click the "Next" button. Supply the names for the fields of Application Name (i.e., ElideTest), Project Name, and Package Name.
3. Use the default settings on the Configure Project setting page, click the "Next" button.
4. Use the default settings, click "Next" from the Configure Launch Icon setting page.
5. Use the default settings, with Create Activity checked and blank activity selected, click "Next" from the "Create Activity" page.
6. Type in the Activity Name (i.e., ElideActivity), Layout Name, and Navigation Type from the "Blank Activity" page. Click "Finish".

Now the new application is created. To add support for native code compiled through the use of the Android NDK, create a "jni" directory under the new project directory. If it is assumed that the project is called "ElideTest", then this new path is called "ElideTest/jni".

3.4 Compile C++ and FORTRAN Dependencies for ELIDe on Android

A major challenge for using C/C++ and FORTRAN together when developing the Android platform is establishing a relationship between the data types that the FORTRAN library uses and understanding which C/C++ data type to use as an argument. If there is a mismatch in the data type using the FORTRAN function, the code may compile without any errors; however, the linker will produce an error. To solve this problem, the proper relationships between the data types used need to be

established. This can be achieved by using the keyword "extern" accompanied by "C." (An example of how this task is accomplished is shown in Appendix A.) Be warned that it is not sufficient to simply use the standard C convention using the keyword "extern" with a function or struct. More details can be seen in an online tutorial.[4]

The additional steps needed to compile the FORTRAN and C++ dependencies for ELIDe are found in Ritchey et al.[1] By following the procedure called out in the Ritchey et al.[1] report, both the libpcap and BLAS dependencies will be compiled for use by ELIDe for Android.

3.5 The Elide Native Code Back End

There are 2 major ways Android applications can communicate with natively compiled applications: call Java Native Interface (JNI) directly and call native executable. These methods are discussed in details in Monitor Network Traffic with Packet Capture (pcap) on Android Device.[5] Both methods must satisfy the root access requirement needed to listen to raw network traffic; however, to perform this task, the native executable was necessary. The libpcap and BLAS were used to accomplish this task, as stated in the previous section.

To utilize libpcap and BLAS, libpcap, BLAS, and "elide" must be on the same source project level. In order for the executable produced by "elide" to use BLAS and libpcap, their compiled libraries needed to be linked. This process was accomplished by modifying the Android project make file (Android.mk) and including the libraries "pcap" and "blas" as well as creating a separate library for the ELIDe functions called "elide_test". The makefile produces a binary called "pcapElide" using the other 3 dependencies as static libraries. An example of the Android.mk file is provided in Appendix B.

3.6 Compile ELIDe Native Executable

The steps to properly compile and build the ELIDe native code are as follows:

- Build the BLAS libraries
- Build the pcap libraries
- Build the ELIDe native code using the "ndk-build" command from the "elide/jni" project directory
- Build the ELIDe C++ header files in the ElideTest directory using the "build-header-script" script

- Build the ELIDe executable by using the "build-script" from the ElideTest/jni directory

3.7 The Android Application Front End

Android applications (apps) are written using the Java programming language. They generally use one or more of the following components as per the Android Developers website[6]:

- Services—A service application is a component that can perform long-running operations in the background and does not provide a user interface. ELIDe for Android makes use of 2 services: ElideTestService and ElideFileStatusService.
- Activities—An activity is a user interface that utilizes the entire screen of a mobile device. An application can utilize multiple activities if it requires multiple full-screen user interfaces. ELIDe for Android makes use of 3 activities: ElideActivity, ViewLogActivity, and ElideSettingActivity.
- Content Providers—Content providers allow for the structure, storage, and security of data within an application. ELIDe for Android uses a single content provider to store the configuration settings for the application.
- Broadcast Receivers—A broadcast receiver is a component that receives notification when a registered event has occurred. ELIDe for Android makes use of 2 broadcast receivers: battery level and service termination.

In addition to the components used by Android applications, ELIDe for Android utilizes a single native-code executable called pcapElide. The Android front end will spawn the pcapElide executable, which performs the data capture and ELIDe analysis natively.

4. Application Operational Breakdown

An Android application requires a main activity to run when the application starts. ElideActivity provides the main user interaction to the ELIDe application. In addition, this activity can be configured to be notified when the available battery power drops by a percentage and record this information along with the current run time of the ElideTestService. This option will help to determine how much power the ELIDe application uses.

In order to turn on the detection capabilities of ELIDe, the ElideTestService initiates the ELIDe native executable pcapElide when the user presses the "Start Service" button in the ElideActivity. This service also initiates the privilege

escalation needed to run the executable with root context. As the pcapElide native executable runs, it 1) records the alerts to a file that the ElideFileStatusService can read, 2) sends Android notifications to the device, and 3) provides the alert statistics to ElideActivity for display. The service is currently configured to retrieve this information every 2 s.

The ViewLogActivity allows users to view the current power-statistics logs, and it can be modified to view other logs such as the alert log. The ElideSettingActivity provides a user interface in which the configuration settings of the application can be modified. This activity is derived from the AndroidPreference Fragment container class so that the information that is stored can be shared among other classes within the application. Diagrams of the ELIDe Android application classes are located in Appendix D.

When the user presses the "Stop Service" button located in the ElideActivity, this will terminate the pcapElide native binary. In addition, if using ELIDe on an offline packet-capture (pcap) file, the JNI module that does not require root escalation to handle ELIDe analysis will terminate once the file has been processed. ElideTestService will then broadcast an intent indicating that ElideTestService has stopped. Because ElideActivity is registered with the ElideTestService, ElideActivity will receive the intent, disable the "Stop Service" button, and enable the "Start Service" button.

5. Elide for Android Application Package

An Android application includes a number of different components, not just the Java source code, and it also includes but is not limited to the following:

- Android Manifest—an XML file that contains the metadata about the application including the application name, required permissions, types of components, component interactions, etc.
- Resources—items such as image files, text information, and user-interface elements are defined here.
- Source—the Java source code for the application.

The AndroidManifest.xml resides in the project's root directory. It is the "must have" file that is generated when the new Android application project is created using Eclipse. In addition, it is an xml file that describes the essential parts of the app. This file contains multiple sections such as the permission section, which indicates the permissions needed that must be granted by the user, and the version

section, which contains information indicating the specific minimum versions of Android required to run the application.

Furthermore, the AndroidManifest.xml file also describes the elements of the Android application including the component name, component type, and component-related elements as well as the interactions among the components via intents. The components described in the file for test application include those listed previously in Section 3.7. An example of an AndroidManifest.xml file is illustrated in Appendix C.

5.1 Deploying the Application

There are 2 methods of deploying the Android ELIDe app. For Method 1, use the "adb install" command on the command console to install the app onto the device. Next, open the shell console and type "adb devices" to make sure the device is connected. Change the directory to the project's bin path, and an .apk file (i.e., ElideTest.apk) should appear. Type the "adb install ElideTest.apk" command on the console, and the app should be installed onto the device. For Method 2, use Eclipse for the deployment: open Eclipse, select the ElideTest project, right click the ElideTest project, and select the "Run As->Android Application" option. This process will install the app to the device.

6. Application Configuration

The ELIDe application for Android has a number of configuration items that can be changed directly from the ElideSettingActivity. Several configuration options were designed to support the research project to measure how much power the Android device utilizes with ELIDe. Initial versions of the program required changing hard-coded configuration options and redeploying the application to the mobile device; therefore, these values became user changeable from the settings activity. The configuration options include the following:

- Notification Interval—changes the delay in which the ElideTestService pulls information from the file generated from pcapElide. The default is 2 s.
- Network Device to Capture—changes the network device to be used for monitoring. The default is wlan0: the primary Wi-Fi network adapter on the device. In addition, none or one offline pcap file on the device can be set.
- Promiscuous Mode—sets the device used for monitoring into promiscuous mode. Typically, this will not change anything because the appropriate

drivers that allow the network device to enable promiscuous mode are not included in a stock Android installation.

- Data Capture Only—activation of ELIDe classification, but it only utilizes libpcap to capture network traffic and move it into a buffer.
- Weight File to Chose From—ELIDe can be trained with different data or at different N-gram lengths or hash lengths. All of this can influence how the weight file is produced. This configuration allows for changing to different weight files.
- pcap File to Be Analyzed—If the network device is set to an offline pcap file, the name of the file can be specified in this configuration option.
- Save Stat Data—By default, ELIDe for Android only saves the network and classification statistics to an alert file after the service has stopped. However, this information can be saved on a regular interval by enabling this option.
- Update Interval—set the update interval for the statistics file.
- Stat File Name to Be Saved—name of the file to save the statistics information.
- Record Battery Levels to File—enables ELIDe for Android to save the current service run time to file every time the battery power drops a percentage.
- Battery Filename to Be Saved—name of the file on which to save battery information.
- Save All Data to pcap File—saves all network traffic to a pcap file.
- pcap File to Be Saved—the name of the pcap file on which to save all traffic.
- Save Alert Data—saves onto a pcap file on which all network-traffic alerts caused by ELIDe analysis.
- Alert pcap File to Be Saved—the name of the pcap file on which to save the alert network traffic.
- Save None Alert Data—saves to a pcap file on which all network traffic that did not cause an alert.
- None Alert pcap File to Be Saved—name of the file on which to store network traffic that did not generate an alert.

All of these options are available from the application during run time. Changes occur once the service is started or restarted. They will not be applied when the service is in capture or classification mode.

7. Application Usage

When starting the ELIDe for Android application, the first activity to greet the user is the ElideActivity. From this screen, the user can start and stop the detection service as well as observe the relevant statistics from the service itself, including the number of alerts generated. In addition, certain network statistics (such as packets captured and dropped) are displayed here. To start or stop the service, the user can press the <start service> button or <stop service> button.

Because the service requires root context to view packets coming into a network interface, a prompt will be presented to the user allowing the program to assume root privileges once the <start service> button is pressed. If the permission is not granted, the service will not start.

Once the ELIDe service is started, all files that are written to the device from within the application are overwritten rather than appended to the same file. For instance, the alert file or statistics file will be cleaned and only updated with the information retrieved during the currently running instance of the service. This information is preserved after service is stopped and only reset after the service is started again. This process will allow data to be retrieved for further analysis without having to reset the files each time the service is run.

If any of the configuration options need to be changed, they can be accessed via the normal Android settings button. Once the <settings> button is pressed, the user will be able to access the ElideSettingActivity. In addition, this process is also used to view the logs or manually refresh the statistics displayed on the main activity page.

7.1 Notifications

While the application is running and the service is performing network-packet classification, the device will receive a notification if an alert has been generated from network traffic. As the alert file produced by the native binary pcapElide is polled, new alerts will generate a notification that will be listed in the Android notification list. The interval to poll can be changed in the configuration settings. Notifications can also be disabled in the configuration settings. If the application is configured to disable ELIDe detection, no notifications will be generated.

8. Conclusions

In this study, ELIDe was found to be effective in performing network packet classification with fewer computational requirements. As a result, this technology has a proven benefit for mobile devices due to its normal battery-usage consumption.[7] This document provides a general guide for any software engineer to write an Android application using ELIDe.

9. References

1. Ritchey RP, Payer GS, Harang RE. Compilation of a network security/machine learning toolchain for android ARM platforms. Adelphi (MD): Army Research Laboratory (US); 2014 Jul. Report No.: ARL-CR-0739.

2. Yu KF. Rooting Android device. Adelphi (MD): Army Research Laboratory (US); 2015 Sep. Report No.: ARL-TN-0706.

3. Yu KF. Android Virtual Machine (VM) Setup on Linux. Adelphi (MD): Army Research Laboratory (US); 2014 Dec. Report No.: ARL-TN-0651.

4. Using C/C++ and Fortran together. YoLinux.com: Linux Tutorials and Information Portal; c1999–2014 [accessed 2014 Dec 03]. http://www.yolinux.com/TUTORIALS/LinuxTutorialMixingFortranAndC.html/.

5. Yu KF. Monitor network traffic with packet capture (pcap) on Android device. Adelphi (MD): Army Research Laboratory (US); 2014 Dec. Report No.: ARL-TN-0650.

6. Android Developers [accessed 2014 Dec 03]. http://developer.android.com/.

7. Payer GS, Yu KF, Harang RE. Characterization of extremely lightweight intrusion detection (ELIDe) power utilization with varying throughput and payload sizes. Adelphi (MD): Army Research Laboratory (US); 2015 Sep. Report No.: ARL-TN-0705.

Appendix A. Example Struct Declaration Using Extern “C”

This appendix appears in its original form, without editorial change.

```
Struct declaration:
```

FORTRAN:

```
      DOUBLE X
      INTEGER Y, Z
      COMMON/XYZ/ X, Y, Z
```

C:

```
      extern struct{
          double x;
          int y, z;
        } xyz_;
```

C++:

```
     extern "C" {
       extern struct{
           double x;
           int y, z;
       } xyz_;
     }
```

Function declaration:

FORTRAN:

```
subroutine fortfunc(a,b)
integer a
real*4  b
```

C:

```
     extern {
       void fortfunc_(int *a, float *b);
       }
```

C++:

```
     extern"C" {
       void fortfunc_(int *a, float *b);
       }
```

Appendix B. Example Android.mk File for ELIDe

This appendix appears in its original form, without editorial change.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
LOCAL_MODULE := blas_LINUX
ifeq "$(TARGET_ARCH_ABI)" "armeabi-v7a"
LOCAL_SRC_FILES :=
$(LOCAL_PATH)/../../blas/obj/local/armeabi-
v7a/libblas_LINUX.a
else ifeq "$(TARGET_ARCH_ABI)" "x86"
LOCAL_SRC_FILES :=
$(LOCAL_PATH)/../../blas/obj/local/x86/libblas_LINUX.a
else
LOCAL_SRC_FILES :=
$(LOCAL_PATH)/../../blas/obj/local/armeabi/libblas_LINUX.a
endif
include $(PREBUILT_STATIC_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := pcap
ifeq "$(TARGET_ARCH_ABI)" "armeabi-v7a"
LOCAL_SRC_FILES :=
$(LOCAL_PATH)/../../pcap_test/obj/local/armeabi-
v7a/libpcap.a
else ifeq "$(TARGET_ARCH_ABI)" "x86"
LOCAL_SRC_FILES :=
$(LOCAL_PATH)/../../pcap/obj/local/x86/libpcap.a
else
LOCAL_SRC_FILES :=
$(LOCAL_PATH)/../../pcap/obj/local/armeabi/libpcap.a
endif
include $(PREBUILT_STATIC_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE    := elide_test
LOCAL_SRC_FILES := MurmurHash3.cpp cmdline_parsing.cpp
snort_oracle.cpp elide.cpp
LOCAL_STATIC_LIBRARIES := blas_LINUX pcap
include $(BUILD_STATIC_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE    := pcapElide
LOCAL_SRC_FILES := pcapElide.cpp
LOCAL_LDLIBS := -llog -landroid
LOCAL_CFLAGS += -std=c++11
LOCAL_STATIC_LIBRARIES := elide_test
include $(BUILD_EXECUTABLE)
```

This scheme basically creates 3 static libraries (blas, pcap, and elide_test) and one executable (pcapElide). This executable can be called by the Android ELIDe application.

INTENTIONALLY LEFT BLANK.

Appendix C. AndroidManifest.xml Example

This appendix appears in its original form, without editorial change.

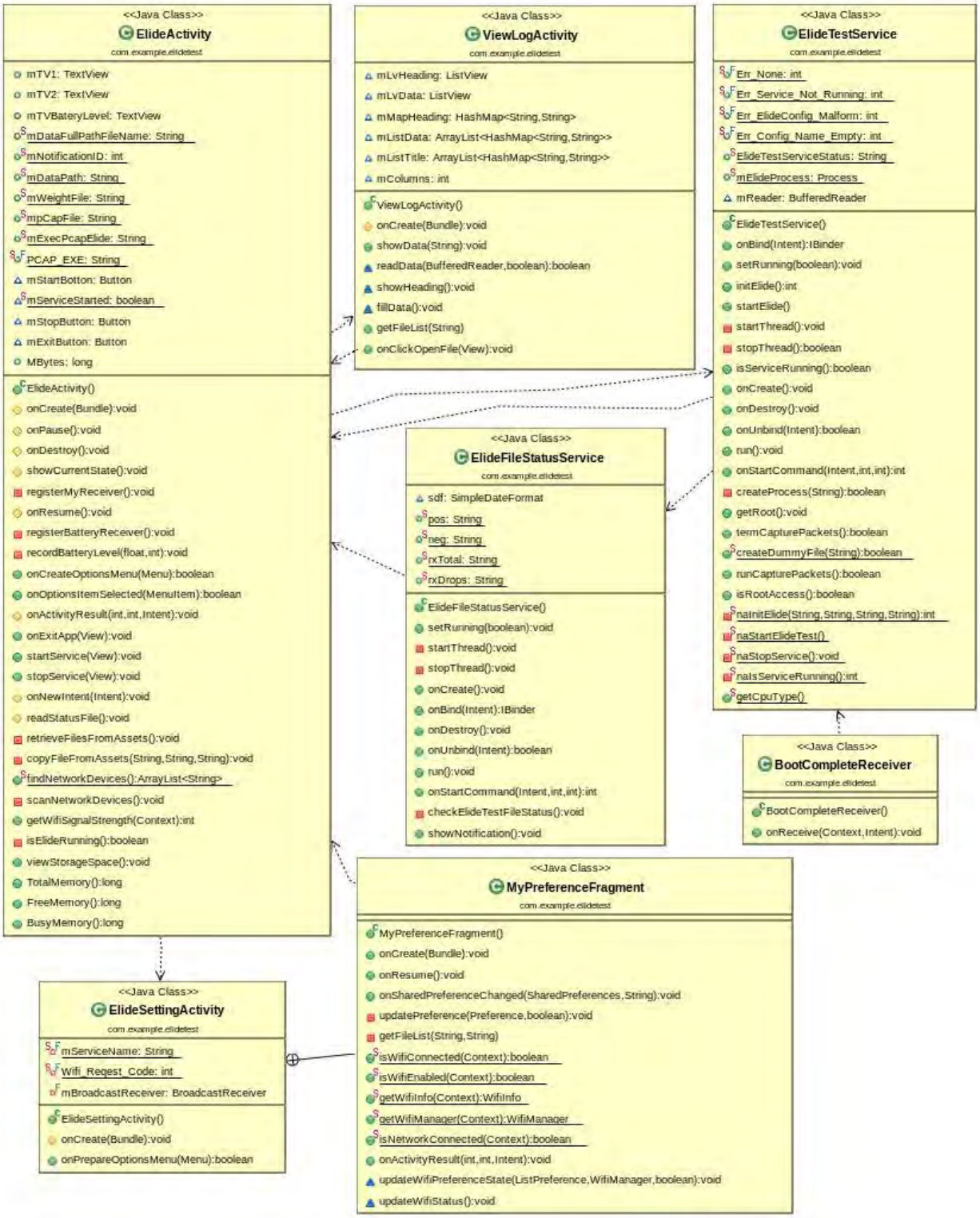
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.elidetest"
    android:versionCode="1"
    android:versionName="1.1" >
    <uses-permission
android:name="android.permission.WRITE_INTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.READ_INTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission
android:name="android.permission.INTERACT_ACROSS_USERS" />
    <uses-permission
android:name="android.permission.VIBRATE" />
    <uses-permission android:required="true"
         android:name="android.permission.INTERNET" />
    <uses-permission android:required="true"
         android:name="android.permission.NETWORK" />
    <uses-permission  android:required="true"

android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission  android:required="true"
        android:name="android.permission.ACCESS_WIFI_STATE"
/>
    <uses-permission  android:required="true"

android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:required="true"

android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity

android:name="com.example.elidetest.ElideActivity"
            android:label="@string/app_name" >
            <intent-filter>
```

```
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".ViewLogActivity" />
        <activity
            android:name=".ElideSettingActivity" />
        <service
android:name="ElideFileStatusService"></service>
        <service android:name="ElideTestService"></service>
        <receiver
android:name="BootCompleteReceiver"></receiver>
            <intent-filter>
                <action
android:name="android.intent.action.BOOT_COMPLETED" />
                <action
android:name="android.intent.action.ACTION_EXTERNAL_APPLICA
TIONS_AVAILABLE" />
                <category
android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        <receiver
android:name="ServiceReceiver"></receiver>
            <intent-filter>
                <action
android:name="com.example.elidetest.ElideActivity.SERVICE_C
OMPLETED" />
                </intent-filter>
            <intent-filter>
                <action
android:name="com.example.elidetest.ElideActivity.STATUS_UP
DATE" />
                </intent-filter>
            </application>

</manifest>
```

INTENTIONALLY LEFT BLANK.

Appendix D. ELIDe Android Application Class Diagrams

This appendix appears in its original form, without editorial change.


<<Java Class>>
ElideActivity
com.example.elidetest
mTV1: TextView
mTV2: TextView
mTVBateryLevel: TextView
mDataFullPathFileName: String
mNotificationID: int
mDataPath: String
mWeightFile: String
mpCapFile: String
mExecPcapElide: String
PCAP_EXE: String
mStartBotton: Button
mServiceStarted: boolean
mStopButton: Button
mExitButton: Button
MBytes: long
ElideActivity()
onCreate(Bundle):void
onPause():void
onDestroy():void
showCurrentState():void
registerMyReceiver():void
onResume():void
registerBatteryReceiver():void
recordBatteryLevel(float,int):void
onCreateOptionsMenu(Menu):boolean
onOptionsItemSelected(MenuItem):boolean
onActivityResult(int,int,Intent):void
onExitApp(View):void
startService(View):void
stopService(View):void
onNewIntent(Intent):void
readStatusFile():void
retrieveFilesFromAssets():void
copyFileFromAssets(String,String,String):void
findNetworkDevices():ArrayList<String>
scanNetworkDevices():void
getWifiSignalStrength(Context):int
isElideRunning():boolean
viewStorageSpace():void
TotalMemory():long
FreeMemory():long
BusyMemory():long
<<Java Class>>
ViewLogActivity
com.example.elidetest
mLvHeading: ListView
mLvData: ListView
mMapHeading: HashMap<String,String>
mListData: ArrayList<HashMap<String,String>>
mListTitle: ArrayList<HashMap<String,String>>
mColumns: int
ViewLogActivity()
onCreate(Bundle):void
showData(String):void
readData(BufferedReader,boolean):boolean
showHeading():void
fillData():void
getFileList(String)
onClickOpenFile(View):void
<<Java Class>>
ElideTestService
com.example.elidetest
Err_None: int
Err_Service_Not_Running: int
Err_ElideConfig_Malform: int
Err_Config_Name_Empty: int
ElideTestServiceStatus: String
mElideProcess: Process
mReader: BufferedReader
ElideTestService()
onBind(Intent):IBinder
setRunning(boolean):void
initElide():int
startElide()
startThread():void
stopThread():boolean
isServiceRunning():boolean
onCreate():void
onDestroy():void
onUnbind(Intent):boolean
run():void
onStartCommand(Intent,int,int):int
createProcess(String):boolean
getRoot():void
termCapturePackets():boolean
createDummyFile(String):boolean
runCapturePackets():boolean
isRootAccess():boolean
naInitElide(String,String,String,String):int
naStartElideTest()
naStopService():void
naIsServiceRunning():int
getCpuType()
<<Java Class>>
ElideFileStatusService
com.example.elidetest
sdf: SimpleDateFormat
pos: String
neg: String
rxTotal: String
rxDrops: String
ElideFileStatusService()
setRunning(boolean):void
startThread():void
stopThread():void
onCreate():void
onBind(Intent):IBinder
onDestroy():void
onUnbind(Intent):boolean
run():void
onStartCommand(Intent,int,int):int
checkElideTestFileStatus():void
showNotification():void
<<Java Class>>
BootCompleteReceiver
com.example.elidetest
BootCompleteReceiver()
onReceive(Context,Intent):void
<<Java Class>>
MyPreferenceFragment
com.example.elidetest
MyPreferenceFragment()
onCreate(Bundle):void
onResume():void
onSharedPreferenceChanged(SharedPreferences,String):void
updatePreference(Preference,boolean):void
getFileList(String,String)
isWifiConnected(Context):boolean
isWifiEnabled(Context):boolean
getWifiInfo(Context):WifiInfo
getWifiManager(Context):WifiManager
isNetworkConnected(Context):boolean
onActivityResult(int,int,Intent):void
updateWifiPreferenceState(ListPreference,WifiManager,boolean):void
updateWifiStatus():void
<<Java Class>>
ElideSettingActivity
com.example.elidetest
mServiceName: String
Wifi_Regest_Code: int
mBroadcastReceiver: BroadcastReceiver
ElideSettingActivity()
onCreate(Bundle):void
onPrepareOptionsMenu(Menu):boolean

List of Symbols, Abbreviations, and Acronyms

AVD	Android Virtual Device
BLAS	Basic Linear Algebra Subprograms
ELIDe	Extremely Lightweight Intrusion Detection
JNI	Java Network Interface
OS	operating system
PC	personal computer
pcap	packet capture
SDK	Software Development Kit

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

2 DIRECTOR
(PDF) US ARMY RSRCH LAB
RDRL CIO LL
RDRL IMAL HRA RECORDS MGMT

2 DIRECTOR
(PDF) US ARMY RSRCH LAB
RDRL CIN D
K YU
G PAYER