Hands-On Exercises for

Android Security Internals

v. 2019.05

WARNING:
The order of the exercises does not always follow the same order of the explanations in the slides. When carrying out the exercises, carefully follow the exercise requirements. Do **NOT** blindly type the commands or code as found in the slides. Read every exercise **in its entirety** before carrying out the instructions.

Class Preparation for HiKey board

1. Install Android Studio:
https://developer.android.com/studio/index.html
This URL should give you access to all the instructions needed to get this installed and working.

2. Install the required packages to build Android -- See the "Installing required packages" instructions for the relevant Ubuntu version here:
https://source.android.com/setup/initializing

3. Fetch the AOSP:
We'll be using 8.1/Oreo. Note that the new lines that start with ">" are printed by the shell. Adding a ">" to the input you type will break the following commands. Assuming you have a directory called "android" in your home directory:

```
$ cd ~/android
$ mkdir aosp-8.1.0_r18
$ cd aosp-8.1.0_r18
$ repo init -u https://android.googlesource.com/platform/manifest \
> -b android-8.1.0_r18
$ repo sync
```

Note that fetching the sources is a fairly lengthy process.

4. Patch 8.1.0_r18 to fix it for HiKey board. These instructions need to be done in the root of the AOSP just downloaded above:

```
$ wget http://opersys.com/downloads/hikey-8.1.0_r18-181008.patch
$ patch -p1 < hikey-8.1.0_r18-181008.patch
```

5. Fetch the binaries for the HiKey board (HDMI support). These instructions are also done in the AOSP root:

```
$ wget \
> https://dl.google.com/dl/android/aosp/linaro-hikey-20170523-4b9ebaff.tgz
$ tar xzf linaro-hikey-20170523-4b9ebaff.tgz
$ ./extract-linaro-hikey.sh
```

You will need to read through the license and type "I ACCEPT" at the end for the binaries to extract.

6. Apply changes for the hands-on exercises:
For the hands-on sessions, we're going to use some custom modifications to the AOSP in

order to add our own instrumentation and measure performance. Note that the new lines that start with ">" are printed by the shell. Adding a ">" to the input you type will break the following commands. Assuming you're still in the "`aosp-8.1.0_r18/`" directory:

```
$ mkdir -p ../orig-files/frameworks-base-services-core-jni/
$ cp device/linaro/hikey/{device-common.mk, init.common.rc, \
> ueventd.common.rc} ../orig-files
$ cp system/sepolicy/private/{file_contexts, hwservice_contexts, \
> service_contexts} ../orig-files
$ cp system/sepolicy/public/init.te ../orig-files
$ cp frameworks/base/Android.mk ../orig-files
$ cp frameworks/base/services/core/jni/{Android.mk, onload.cpp} \
> ../orig-files/frameworks-base-services-core-jni/
$ cp frameworks/base/services/java/com/android/server/SystemServer.java \
> ../orig-files/
$ cd ..
$ wget \
> http://opersys.com/downloads/libhw-opersys-hikey-8.1-full-180912.tar.bz2
$ tar xvjf libhw-opersys-hikey-8.1-full-180912.tar.bz2
$ find libhw-opersys-hikey-8.1-full-180912 -exec touch {} \;
$ cp -r libhw-opersys-hikey-8.1-full-180912/* aosp-8.1.0_r18
```

7. Make sure you have "mtools" installed:

```
    $ sudo apt-get install mtools
```

8. Build the AOSP:

```
    $ cd ~/android/aosp-8.1.0_r18
    $ . build/envsetup.sh
    $ lunch hikey-eng
    $ make -j8
```

9. Ensure that the build finishes successfully.

10. Flash your board with your AOSP:
    (a) Plug your board into the power jack and your computer through the USB cable. The board already has a system image on it that it'll boot with.
    (b) Make sure your workstation recognizes the device and gives you access to it. Have a look at the instructions here for more information: https://developer.android.com/studio/run/device.html#setting-up
    (c) Put the board in "fastboot" mode:

```
    $ cd ~/android/aosp-8.1.0_r34
    $ . build/envsetup.sh
    $ lunch hikey-eng
    $ adb reboot bootloader
```

(d) The board should reboot and you should be able to see it using this command – example output provided:
```
$ fastboot devices
76B8809900099B63  fastboot
```
(e) Reflash with your own images:
```
$ fastboot flashall
target reported max download size of 134217728 bytes
--------------------------------------------
Bootloader Version...: 0.4
Baseband Version.....: 0.4
Serial Number........: 76B8809900099B63
--------------------------------------------
checking product...
OKAY [  0.001s]
checking version-bootloader...
OKAY [  0.001s]
sending 'boot' (24380 KB)...
OKAY [  0.818s]
writing 'boot'...
...
OKAY [  3.097s]
writing 'system' 7/7...
OKAY [  3.819s]
rebooting...

finished. total time: 66.887s
```
(f) Check that your device booted properly into Android mode:
```
$ adb devices
List of devices attached
76B8809900099B63  device
```
(g) Log into your device:
```
$ adb shell
hikey:/ #
```
(h) Check that the build description matches your build date/time:
```
hikey:/ # cat system/build.prop | grep -i description
# Do not try to parse description, fingerprint, or thumbprint
ro.build.description=hikey-eng 8.1.0 OPM5.171019.017 eng.karim.20180308.124948 test-keys
```

SEAndroid/SELinux

1. Familiarize yourself with the following tools and commands if you've never used the before:
   - getenforce
   - ps -a -Z
   - ls -al -Z
   - id -Z

- dmesg | grep avc

2. Try the “audit2allow” tool on the host to obtain SELinux rules that fix errors reported by grepping the kernel dmesg output for denial errors.

3. Explore the following filesystem entries to discover the relevant SELinux kernel capabilities exposed to user-space:
- /sys/fs/selinux/
- /sys/fs/selinux/class and its subdirectories and files
- /proc/PID/attr

4) Get the client and server programs from http://www.linuxhowtos.org/C_C++/socket.htm. Place both in device/linaro/hikey and add the relevant Android.mk files to compile them in your AOSP. For example, to compile the C server:
- Add it as [aosp]/device/linaro/hikey example-server
- You may need to add “#include <strings.h>” to the C file for it to build
- Make sure “server” is added to [aosp]/device/linaro/hikey/device-common.mk
- Make sure you have an appropriate Android.mk within [aosp]/device/linaro/hikey/example-server to get your app to build. Here's a sample:

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

LOCAL_SRC_FILES:= \
        server.c

LOCAL_SHARED_LIBRARIES := \
        libcutils

LOCAL_MODULE:= example-server

include $(BUILD_EXECUTABLE)
```

5) Add an .rc file to the server build to have it started by init at startup.

6) Add relevant rules in device/linaro/hikey/sepolicy to:
- Define example_server and example_client domains
- Use an domain_auto_trans() rule (or better yet, an init_daemon_domain()) to transition from init to example_server when the server is run.
- Enable example_server and example_client domains to communicate over sockets

7) Modify the server program to use libselinux to retrieve the client's SE context and print it out

when client connects.

8) Modify the circular character driver in device/linaro/hikey/circular-driver to print the security context SID of the process opening it. Have a look at:

- struct file's f_security field
- security/selinux/hooks.c
- file_security_struct in security/selinux/include/objsec.h

9) (optional) There's a security_sid_to_context() in security/selinux/include/security.h and security/selinux/ss/services.c, but it's not exported for modules to use. Try exposing it using EXPORT_SYMBOL() and use it in your module to convert the SID to a string context and print it out.

10) Modify the rules for the Opersys system service components to "fix" them in system/sepolicy/. Look for "opersys" and "circchar" entries using grep. Currently, they're all "patched" to just work. You'll need to define entries such as:

- opersys_device
- opersys_service
- hal_opersys_hwservice

11) Modify Opersys stack (see the libhw-opersys-hikey-8.1-full-180912.tar.bz2 contents downloaded above) to start as separate platform-signed persistent APK with its own user ID:

- put under top-level "device/linaro/hikey/" directory
- android:persistent = true (see packages/services/Telephony)
- Shared USER ID == one in the OEM range (look for "OEM" in system/core/include/cutils/android_filesystem_config.h)

12) Change SELinux rules to match the persistent app just added instead of a built-in system service as in #10

- Move all non-hidl definitions to device/linaro/hikey/sepolicy files
- Use seapp_contexts to set domain based on user ID and app package name

AOSP User-Space

1) Generate a pair of adb keys using “adb keygen”

2) Create your own cert for signing the system service APK created in the previous exercises and integrate that new cert into your AOSP build and SELinux rules. There is some “googling” involved here.

3) Generate your own release keys for slide 305

4) Generate a full OTA zip file (see slide 307)

5) Make a modification to your AOSP (your choice) and rebuild the AOSP

6) Generate an incremental OTA zip file (see slide 308)