ABPS Documentation for Testers and Developers

Matteo Martelli

December 16, 2016

This document describes the steps and actions required for building all the ABPS software parts I have implemented. They form essentially an early "documentation for testers and developers" whose intent is to provide a guideline for deploying the same test scenarios I used in my analysis phase and for setting up an implementation starting point for future developers.

At the time of writing this documentation is maintained on my personal ABPS github repository: *https://github.com/matteomartelli/ABPS*. Often in the next sections it will be used the terms "this repository" or the "abps repo" in reference of such repository. Thus the first required step is to clone the repository since it contains all the software tools which this documentation refers to:

```
git clone https://github.com/matteomartelli/ABPS.git
```

1 TED kernel and proxy application

This section explains how to to build a custom kernel with the support for TED (Transmission Error Detector) and how to use it with the TED proxy application. Different build methods are needed whether you want to build the TED kernel for a GNU Linux distribution or Android distribution, since some additional steps are required for the latter.

1.1 Build Linux kernel

1.1.1 Get kernel sources

Clone the linux kernel repository in your local machine.

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
    linuxrepo
```

Then move to the version branch you want to build. TED patches are currently available for versions 4.1.0 and 3.4.0 but you can always move to a different version branch and manually adjust the TED patch. Let's assume you chose the 4.1.0 version.

```
cd linuxrepo
git checkout v4.1
```

In order to be sure your git head is at the desired version, just check the first 3 lines of the Makefile located in the repository root directory.

1.1.2 Patch the kernel

If you chose the version 3.4.0 of the kernel you should first apply an official patch needed by TED to work properly. This patch was introduced in later versions and allows TED to read some information about its socket at lower levels of the network stack.

From the root directory of the linux kernel repository:

```
patch -p1 < abps_repo/ted_proxy/kernel_patches/net-remove-skb_orphan_try.3.4.5.
    patch
```

Then apply the TED patch. For the 4.1.0 version only this step is needed to patch the kernel sources.

```
patch -p1 < abps_repo/ted_proxy/kernel_patches/ted_linux_(your_chosen_version).
    patch
```

1.1.3 Build kernel

Several well documented guides explaining how to build a custom kernel exists in the web[6][5].

As guide [6] explains, `make localmodconf` is recommended for faster compilation but be sure that all the modules you will need later are currently loaded. At the end, just run to start the build:

```
make -jN
```

(where N is the number of parallel compilation processes you want to spawn).

After your custom kernel is built just run with root privileges:

```
make modules_install
make install
```

or refer to your linux distribution kernel install method.

1.2 Android

The following steps are known to be working, at the time of writing, for the LG Nexus 5 with Android 6 Marshmallow.

1.2.1 Prerequisites

You will need some Android tools such as `adb` and `fastboot`. In debian like distributions and Arch linux distributions they should be available through the `android-tools` package. Also you need the "Android NDK toolset". This toolset will be used to compile the `tedproxy` application and can be downloaded from the Android developers reference website[1].

1.2.2 Enable root privileges

Since TED requires a device which supports mac80211, the inner WiFi module (Broadcom BCM4329) of the LG Nexus 5 can't be used. In fact the BCM4329 module works as a Full MAC device with a proprietary and closed source firmware, without any support for the mac80211 subsystem. Anyway the TED kernel and its proxy application can be tested using an external USB WiFi dongle which supports the mac80211 driver interface. To do so, some Android system configuration files must be edited with root privileges. Also, the `tedproxy` application binds the sockets directly to the network interfaces through the socket option `SO_BINDTODEVICE` which requires root privileges. Root tools for Android, essentially allow users to execute the `su` binary file which is missing by default for security purposes.

The tool I used is Superuser[8]. It is open source and offers both the boot image that includes the su binary file and the permissions control application. The installation procedure for the Nexus 5 is quite simple since the superuser community already provides pre-built boot images at `https://superuser.phh.me`.

Once you have obtained the boot image for your device, first you have to unlock the bootloader, then simply flash the boot image with the fastboot tool (it may damage your device):

```
fastboot oem unlock
fastboot boot nameofrecovery.img
```

You can also use a custom recovery such as TWRP[9] which allows you to backup the original boot image first.

If a pre-built image for your device is not available you should execute the content of the superuser.zip installation file in a custom recovery such as TWRP since it can run scripts with root privileges. In brief, if you "flash" the zip file in the TWRP recovery, it extract the zip file and executes the extracted scripts. These scripts essentially copy the current boot partition in a boot image file, extract the boot image, extract the inner ramdisk image and copy the su binary file into the extracted ramdisk. Lastly a modified boot image will be re-created and actually flashed to the device.

1.2.3 Get tools and kernel sources

First take a look at the official android documentation[4] in order to understand which kernel version you need depending on your device and get the right tools. These are the steps needed to get the right tools and kernel version for a LG Nexus 5 device.

Get the pre-built toolchain (compiler, liker, etc..), move it wherever you like and export its path:

```
git clone https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/arm/
    arm-eabi-4.6
sudo mv arm-eabi-4.6 /opt/
export PATH=/opt/arm-eabi-4.6/bin:\$PATH
```

You can also copy the last line inside your .bashrc.

Get the kernel sources:

```
git clone https://android.googlesource.com/kernel/msm
git checkout android-msm-hammerhead-3.4-marshmallow-mr2
```

1.3 Patch the kernel

From the root directory of the android kernel repository:

```
patch -p1 < abps_repo/ted_proxy/kernel_patches/net-remove-skb_orphan_try.3.4.5.
    patch
patch -p1 < abps_repo/ted_proxy/kernel_patches/ted_linux_3.4.patch
```

1.3.1 Configure and build

In order to test TED with the Nexus 5 you should add in your custom kernel the support for a USB Wi-Fi dongle that is mac80211 capable. This repo provide a custom configuration which add the support for the Atheros ath9k_htc and Ralink rt2800. You can of course make your own configuration to add the support for different devices.

First of all let's prepare the environment. From the root directory of the android kernel repository:

```
export ARCH=arm
export SUBARCH=arm
export CROSS_COMPILE=arm-eabi-
```

Then let's make the configuration. If you are fine with my custom configuration:

```
cp abps_repo/tedproxy/android_build/hammerhead_defconfig_ted .config
```

Otherwise if you want to make your own configuration:

```
cp arch/arm/configs/hammerhead_deconfig .config
make menuconfig
```

Then start the build:

```
make -jN
```

(where N is the number of parallel compilation processes you want to spawn).

1.3.2 Common issues

Compiling old kernels from a new linux distribution may require some workaround.

If you encounter an error on scripts/gcc-wrapper.py, it may be caused by the fact that your default python binary links to python3 while that script wants python2.

A simple workaround consists in replacing the first line of scripts/gcc-wrapper.py with:

```
#! /usr/bin/env python2
```

Another error that you may encounter is

```
Can't use 'defined(@array) (Maybe you should just omit the defined()?) at kernel/
    timeconst.pl
```

To avoid this just remove all the defined() invocation, without removing the inner array variables, from kernel/timeconst.pl as the comment in the error suggests.

Once the build is finished, the kernel image is located at arch/arm/boot/zImage-dtb.

1.3.3 Enable the USB Wi-Fi Dongle

As many Android devices, the Nexus 5 boot process is handled by an init script contained in the ramdisk filesystem image, which is itself contained in the boot image together with the kernel image.

The init script is the one who starts the `wpa_supplicant` daemon with the default interface `wlan0`. I had to modify the init script inside the ramdisk image in order to start `wpa_supplicant` with the external usb dongle, which is `wlan1`. To do so, you'd need to extract the original boot image from your device, extract the ramdisk image, modify the `init.hammerhead.rc` file substituting all the `wlan0` with `wlan1`.

This repository already provides a modified ramdisk image, anyway these are the steps you'd need to follow in order to retrieve your original boot image from the Nexus 5:

Get the original boot.img

```
#use the following commands to find the boot partition
ls -l /dev/block/platform/
#now we know the device platform is msm_sdcc.1
ls -l /dev/block/platform/msm_sdcc.1/by-name
#now we know the boot partition is mmcblk0p19
#by [boot -> /dev/block/mmcblk0p19]
#use the following command to retrieve the boot.img
su
cat /dev/block/mmcblk0p19 > \
    /sdcard/boot-from-android-device.img
chmod 0666 /sdcard/boot-from-android-device.img
```

You can then copy the image to your machine with `adb pull` or the MTP protocol.

Extract the original boot.img Once you have your original boot image in your hand, you can proceed with the extraction.

I recommend to read this guide[3] and use this tool[2] for the boot image extraction. The latter is also mirrored in this repository also under `abps_repo/tedproxy/android_build/boot/boot-extract`.

```
./boot-extract boot.img
```

Store the output of the extraction, since it will be necessary later for the boot image re-creation. In my case this is the output:

```
Boot header
  flash page size       2048
  kernel size           0x86e968
  kernel load addr      0x8000
  ramdisk size          0x12dab8
  ramdisk load addr     0x2900000
  second size           0x0
  second load addr      0xf00000
  tags addr             0x2700000
  product name          ''
  kernel cmdline        'console=ttyHSL0,115200,n8 androidboot.hardware=
      hammerhead user_debug=31 maxcpus=2 msm_watchdog_v2.enable=1'

zImage extracted
ramdisk offset 8845312 (0x86f800)
ramdisk.cpio.gz extracted
```

Extract and edit the original ramdisk Once you have your ramdisk image `ramdisk.cpio.gz` you can extract it with:

```
mkdir ramdisk_dir
cd ramdisk_dir
zcat ../ramdisk.cpio.gz | cpio -i
```

Finally you can edit the `init.hammerhead.rc` file substituting all the `wlan0` with `wlan1`.

Create the custom ramdisk After that, re-create the ramdisk filesystem image with the `mkbootfs` tool:

```
cd ..
abps_repo/tedproxy/android_build/boot/mkbootfs/mkbootfs ramdisk_dir > ramdisk.ted
    .cpio
gzip ramdisk.ted.cpio
```

The result is a modified ramdisk image: `ramdisk.ted.cpio.gz`. If this fails to boot you can try with the pre-made custom ramdisk available at `path_to_abps_repo/tedproxy/android_build/boot/ramdisk.ted.cpio.gz`.

Create the custom boot.img Now re-create the boot image from both the custom kernel and the custom ramdisk image:

```
abps_repo/tedproxy/android_build/boot/mkbootimg/mkbootimg --base 0 --pagesize
    2048 --kernel_offset 0x00008000 \
--ramdisk_offset 0x02900000 --second_offset 0x00f00000 --tags_offset 0x02700000 \
--cmdline 'console=ttyHSL0,115200,n8 androidboot.hardware=hammerhead user_debug
    =31 maxcpus=2 msm_watchdog_v2.enable=1' \
--kernel path_to_kernel/zImage-dtb --ramdisk path_to_ramdisk/ramdisk.ted.cpio.gz
    -o boot.img
```

Note how the offsets correspond to the addresses printed out by the boot image extract script.

Enable wlan1 in system files Editing the init.rc file is not enough, as some other Android services still refer to the wlan0 device.

You need also to substitute `wlan0` with `wlan1` in `/system/build.prop` and `/system/etc/dhcpcd/dhcpcd.conf`. These files are persistent in the Android system partition thus you just need to edit them once.

Also you need to copy the firmware of your WiFi dongle in /system/etc/firmware. You can copy directly from your working machine or from the official repository[7].

At the end you can boot the custom boot.img with fastboot:

```
adb reboot bootloader
sudo fastboot boot boot.img
```

Or if you are sure of what you are doing you can flash it:

```
sudo fastboot flash boot boot.img
```

Once the Nexus rebooted it can be useful to activate the remote adb access:

```
adb tcpip 5555
```

Then you can plug the external Wi-Fi dongle with an OTG cable and control the device with remote adb:

```
adb connect nexus_ip_address:5555
```

Open issue The sleep mode of the external Wi-Fi dongle is not handled properly. In fact turning off the LCD screen cause the Wi-Fi device to de-associate from the network. As a simple workaround you can use an Android App to force your screen active but consider that this approach will lead your device to rapidly exhaust its battery power.

2 Build and run `tedproxy`

2.1 Build

Before you can build the `tedproxy` application you must ensure you are running a TED custom kernel. Then some header files called "user api (or uapi)" must be modified. These headers simply contain declarations of kernel constants and macros that also the user applications may need to recall. If you want to build the `tedproxy` application for linux distributions:

```
su
cp /usr/include/linux/errqueue.h \
    /usr/include/linux/errqueue.h.bkp
cp /usr/include/linux/socket.h \
    /usr/include/linux/socket.h.bkp
cp abps_repo/tedproxy/uapi/errqueue.h \
    /usr/include/linux/errqueue.h
cp abps_repo/tedproxy/uapi/socket.h \
    /usr/include/linux/socket.h
```

Otherwise if you want to build the `tedproxy` application for Android:

```
cd path_to_ndk/platforms/android-21/arch-arm/usr/include/linux
cp errqueue.h errqueue.h.bkp ; cp socket.h socket.h.bkp
cp abps_repo/tedproxy/uapi/errqueue.h errqueue.h
cp abps_repo/tedproxy/uapi/socket.h socket.h
```

At the end you just need to run the `build.sh` script with `linux` or `android` as argument for building the respective versions of teproxy. The build.sh is at `path_to_abps_repo/tedproxy/tedproxy`.

The Android version binary file will be placed at `libs/jni/tedproxy` and you can copy it on your device with adb push.

2.2 Run

`tedproxy` is intended to work as a local proxy which listens UDP traffic from a local bound socket and forwards all the input packets to the remote host through one or multiple network interfaces binding a socket for each one. With the '-i' option you can specify the name of the chosen network interfaces. If one of them is a WiFi mac80211 capable device, you can put the 't:' prefix before the device name and tproxy will enable TED notification for that device. In this case, packets will be forwarded to the "TED interface" only as long as the number of packets received at the AP is sufficiently high. Whenever this condition is no longer satisfied, `tedproxy` also enables the other network interfaces for forwarding traffic.

For instance `tedproxy` can listen to UDP local port 5006 and forward everything to host 130.136.4.138 at UDP port 5001 through the wlan1 mac80211 device and the rmnet0 device which is usually the cellular network interface on Android smartphones:

```
tedproxy -b -i t:wlan1 -i rmnet0 5006 130.136.4.138 5001
```

3 Relay and CN tools

3.1 Relay

The relay activation is quite simple. From the proxy server that you elected as the RTP relay do the following:

```
cd abps_repo/udprelay
python3 udprelay.py 5001:5002
```

Start the udprelay process which listens on two UDP ports. Indeed you can specify any ports you like but keep in mind the first is where the MN attaches to and the second is where the fixed CN attaches to.

3.2 CN tools

CN tools is a set of software tools collected in `abps_repo/cntools` that are required to enable an RTP receiving end on the CN. First of all you need `ffmpeg` installed on the CN machine. At the time of writing `ffmpeg` is not present on debian distributions since it has been replaced with `avconv`. I did not investigate further but `avconv` did not work well in my test environment. Thus since my CN was running a debian "jessie" distribution I compiled `ffmpeg` from sources. Once you obtained `ffmpeg` working on your CN machine you can run the `launcher.sh` script:

```
cd abps_repo/cntools
./launcher.sh 130.136.4.138 5002 5006 outputvideo.ogg
```

where 130.136.4.138 is the remote host IP address, 5002 the remote UDP port and 5006 the local UDP port.

The script first launches the `dummystdserver` HTTP server which serves an SDP file at the address `http://127.0.0.1:8080/camera.sdp`. Then it launches `cnproxy` passing to it the host IP address and both ports as arguments and runs `ffmpeg` as last step. When launched `cnproxy` sends an initialization datagram to the remote host then starts forwarding remote host's responses to the local UDP port, 5006 in this case, which `ffmpeg` listens to. Thus `ffmpeg` first gets the `camera.sdp` file from the `dummystdserver` then receives on a local UDP port, specified in the sdp file, datagrams forwarded by `cnproxy` and eventually writes the output video file. Be sure the remote UDP port corresponds to the one the destination is listening to and that the local UDP port corresponds to the one specified in the sdp file.

3.3 Put everything together

Now that you have everything installed and configured you can stream some UDP traffic from the multi-homed mobile device to the CN passing through the relay. For instance let us consider an RTP camera streamer installed on the mobile device that sends RTP packets to the CN. On the mobile device you must first run `tedproxy`:

```
tedproxy -b -i t:wlan1 -i rmnet0 5006 130.136.4.138 5001
```

then activate the relay application on the remote proxy:

```
python3 udprelay.py 5001:5002
```

Now start the CN tools with the `launcher.sh` script:

```
./launcher.sh 130.136.4.138 5002 5006 outputvideo.ogg
```

It will send an initialization packet to the relay and then it will wait for user confirmation before starting `ffmpeg`. Before confirming, start the stream from the mobile device. Be sure the RTP streamer application sends packets to 127.0.0.1 and UDP port 5006 which is the port `tedproxy` is listening to. I used a proprietary camera streamer application (`https://play.google.com/store/apps/details?id=com.miv.rtpcamera`). However any similar application should work well.

References

[1] Android ndk downloads page. https://developer.android.com/ndk/downloads/index.html.

[2] boot-extract. https://github.com/csimmonds/boot-extract.

[3] Booting android. http://www.slideshare.net/chrissimmonds/android-bootslides20.

[4] Building kernels. http://web.archive.org/web/20161021065948/http://source.android.com/source/building-kernels.html.

[5] *KernelBuild*. http://web.archive.org/web/20160825174928/https://kernelnewbies.org/KernelBuild.

[6] *Kernels/Traditional compilation*. http://web.archive.org/web/20161107160059/https://wiki.archlinux.org/index.php/Kernels/Traditional_compilation.

[7] linux firmware. http://git.kernel.org/cgit/linux/kernel/git/firmware/linux-firmware.git.

[8] Superuser. https://github.com/koush/Superuser.

[9] Twrp. https://twrp.me/about.