Krzysztof Adamski

Alcatel One Touch Fire Hacking (Mini)Guide

Introduction

Tue 24 September 2013
By Krzysztof Adamski
In Log.
tags: firefox-os one-touch-fire

`Alcatel One Touch Fire` is one of first two production phones running `Firefox OS`. Unlike `ZTE Open`, I haven't seen too much information on how to make it hacker friendly. This is why I created this short guide.

First thing you want to do with your phone when doing some Firefox OS development is to flash your custom firmware built. Currently the phone is running the only stable release of the system which is `1.0.1`. Because of the way Mozilla does its development, it feels like it's ancient and the first thing you want to do with the phone when you get it is to upgrade the software. After recently fixed bug, at least version 1.1 should be working OK. Let's see how can safely try it.

Keep in mind that following my guide you may brick your device. Proceed at your own risk.

Backing up the original firmware

Before changing anything on the phone, lets first backup original firmware so we can get back to it when something goes wrong. Unfortunately this process is a little bit more complicated than it should. First thing we have to do is to get root privileges and we want to do this without changing anything on the device itself.

Step 1: Build B2G

We will start from building `B2G` - follow this guide. Use `BRANCH=v1-train` (which is default as of this writing). This will take a lot of time and will require quite a lot of RAM. While waiting for this to finish, you can prepare to next steps.

Step 2: Create custom kernel

Download and build kernel sources from sourceforge.net. They are released by Alcatel but for some reason, they doesn't work too well. It is supposed to be the same source code that is used for producing kernel for devices that are sold. Unfortunately It seems it's not the case. When using this kernel, you will see a lot of glitches on the screen and some errors like this:

```
<3>[   36.310903] mdp_ppp: could not retrieve image from memory
```

in dmesg. This problem can be fixed by applying this patch. But then again, the kernel crashes when you make a phone call. Fortunately we only need this temporarily in order to get actual kernel binary from the device. It would be great if we could compile fully working kernel ourselves but I can't do this right now.

In order to compile the kernel, I used precompiled toolchain from B2G's Android tree. Something like:

```
$ cd B2G/
$ export PATH=${PWD}/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/:${PATH}
$ cd ONE_TOUCH_FIRE_4012_20130628/kernel/
```

```
$ make CROSS_COMPILE=arm-eabi- ARCH=arm zImage
```

When this finishes, you can find your kernel binary in `kernel/arch/arm/boot/zImage` file.

Step 3: Create custom boot.img

Using kernel built in Step 2 and `ramdisk.img` built in Step 1, we'll create `boot.img` we could boot our device from. You can find your `ramdisk.img` in `out/target/product/hamachi/` folder. You will also need `mkbootimg` tool which can be found in `out/host/linux-x86/bin/mkbootimg`:

```
$ export PATH=${PWD}/out/host/linux-x86/bin/:${PATH}
$ mkbootimg --kernel ONE_TOUCH_FIRE_4012_20130628/kernel/kernel/arch/arm/boot/zImage \
--ramdisk out/target/product/hamachi/ramdisk.img --base 0x200000 --cmdline \
'androidboot.hardware=qcom loglevel=1' -o myboot.img
```

Step 4: Boot your myboot.img

To do anything with the bootloader, you need `fastboot` tool. It can be found in Android SDK package, in `platform-tools` directory. It may also be packaged for you distribution (`android-tools` on Fedora, for example).

Start your device in `fastboot` mode by holding `volume down` button just after powering it up. It will stop at initial Alcatel's logo. You can confirm that it is connected by typing `fastboot devices`. You should see something like:

```
$ fastboot devices
MSM7627A        fastboot
```

Note that by default in most Linux distributions normal users wont be able to see any fastboot devices. You can either use `sudo` to run `fastboot` or configure your `udev` to grant your user read-write access to the device. Something like this should do the trick:

```
$ echo 'SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="d00d", MODE="0666", OWNER="{YOURUSER}"
$ sudo systemctl restart systemd-udevd.service
```

While at this, you could also add similar rule for `05c6:9025` which is ID used by the phone in normal (non-fastboot) mode. It will be handy for using `adb` in the future.

Now that our device is in fastboot mode, we can boot it with our own kernel/initramfs created in Step 3 with:

```
$ fastboot boot myboot.img
```

Step 5: Create backup images

Now start shell on the phone using `adb shell` command and dump backup images to SD card:

```
$ cat /dev/mtd/mtd0 >/mnt/sdcard/boot.img
$ cat /dev/mtd/mtd1 >/mnt/sdcard/system.img
$ cat /dev/mtd/mtd5 >/mnt/sdcard/userdata.img
$ cat /dev/mtd/mtd7 >/mnt/sdcard/recovery.img
```

When finished (you may also dump some more MTD partitions if you like), transfer those files to your computer. You may again use `ADB` for that:

```
$ mkdir hamachi-backup
$ I="boot.img system.img userdata.img recovery.img"
$ for i in $I; do adb pull /mnt/sdcard/$i hamachi-backup/; done
```

Step 6: Recreate original boot.img

Now that we have original `boot.img` we can easily change what's most important for us in this image - `ro.secure`

setting. Which user `adbd` is running at depends on this setting so we want it to be set to `0` instead of `1`. We need a tool to "unpack" the image. There are couple of them available, but I used unbootimg as it's build with `B2G` source. You can find the binary in `out/host/linux-x86/obj/EXECUTABLES/unbootimg_intermediates/unbootimg`. Here's how it can be used:

```
$ PATH=$PATH:out/host/linux-x86/obj/EXECUTABLES/unbootimg_intermediates/
$ cd hamachi-backup/
$ ../external/unbootimg/unpack.sh boot.img
```

Edit `boot.img-ramdisk/default.prop` file and set `ro.secure=0`. Then rebuild `boot.img`:

```
$ cp boot.img{,.original}
$ ../external/unbootimg/repack.sh boot.img
```

Profit

Now you have new `boot.img` with exactly the same kernel image that is used in production firmware. You can boot (the same way we did this in Step 4) or you can flash it using `fastboot` for permanent change. Either way you should get root with `adb shell` now. This means you should be able to use all kinds flavours of B2G's `./flash.sh` command to upgrade the system. You also have backup images so as long as you don't damage `fastboot` on the device, you should be able to recover to the stock image.