Brandi Crawford

Donald Raikes

CS9223: Mobile Security

Spring 2014

Team Project

NFC Simulation Lab

Since neither Brandi nor I has an NFC-enabled smartphone, we decided it would be most effective to attempt to use the open-nfc project’s nfc simulator to conduct some experiments with nfc. The simulator works with an android virtual device to simulate reading nfc tags and/or performing peer-to-peer communication between two android devices. On the surface of it, it looked like it was going to be a “simple” setup, but things are not as they seem.

Having had some experience running android virtual devices (avds) before, I took the lead on this part of the project. After doing a lot of reading, I found an article that described how to get started working with the nfc simulator (<http://stackoverflow.com/questions/6770774/getting-started-with-open-nfc-emulator>). This contained a lot of really good information, so I followed the instructions and got … nothing.

The simulation needs to run on windows since the nfcc-simulator is a windows executable. I am running windows7 64-bit so that was no problem. I downloaded the latest version of the open-nfc android add-on (version 4.5.2), the latest version of the android development tool bundle for windows from the developer.android.com website which includes eclipse and the latest version of the android sdk. I also downloaded the echotool although I never quite figured out why it was necessary to the setup.

So far so good. The instructions say next to extract the open-nfc android add-on zip archive and the adt bundle zip archive, so I did that. I ran the SDK manager application from the ADT bundle and downloaded images for android versions 4.0.3 through 4.4.1 (icecream sandwich through kitkat) along with their development tools and extras. So the sdk was all setup.

Next, I copied the add-ons folder from the open-nfc android add-on bundle and pasted it to the sdk’s add-ons folder. This gave me an add-on labeled “addon-2013-02-28\_android-4.2.1\_r1\_OpenNFC\_4.5.2” which according to the documentation should allow me to create an avd based on that custom android rom. Glitch number 1: when I opened the android virtual device manager, I couldn’t see the open-nfc rom anywhere, so back to google. I found a link to a page that described how to test custom android roms with the sdk and avd manager (<http://forum.xda-developers.com/showthread.php?t=906161>). Basically I needed to create an avd with the same version of android that my custom rom was based on and drop the .img files into the folder containing my avd’s files (c:\users\draikes\.android\avvd\<avd-name>). So I did that and after customizing the avd for accessibility (not normally necessary, but for me it is), I launched thte emulator to run the avd. Voila, I had a functional android 4.1.2 avd with the open-nfc stack included.

Next step was to test the simulator piece. I had to start the open-nfc ConnectionCenter ( all connections between avds and the simulator go through the ConnectionCenter), and run the nfcc-simulator as administrator. This worked just fine. I configured the ConnectionCenter to accept connections from other machines as the instructions said to do and all seemed good. However, when I went to the avd and clicked on settings wireless-networks, I could not check the box to enable nfc. My first thought was that the .img files from the open-nfc download were somehow corrupted, so I downloaded all the necessary pieces on my linux laptop and following the open-nfc porting guide, I built my own copy of the open-nfc enabled android rom. I dropped the new .img files in place, and the same thing happened I was unable to check the enable nfc checkbox. I went back to google again. Eventually I found a page that said that there should be an actual app listed under the apps in the emulator called open-nfc settings. This app would allow me to enable nfc and set all the necessary parameters to talk with the simulator. The app was missing from the avd.

Since the open-nfc mailing lists seemed to be defunct, I had to get creative. My solution was to try earlier versions of the open-nfc android add-on packages. I tried version 4.5.1, 4.5.0, and finally 4.4.1. After downloading each bundle and extracting it, I would copy the .img files into my avd make sure the accessibility features were enabled and test it out. Finally in version 4.4.1, I found the open-nfc settings app!

Now the fun began. I started the ConnectionCenter, and the nfcc-simulator. I started the echotool and launched the avd. An additional tool that comes with the connectioncenter is a trace logger wich logs tons of data aobut everything happening in my system. That was running in the background as well. I switched to the simulator selected one of the 50 available virtual nfc tags and double-clicked it. That “presents it” to the antenna of the android avd that is running. The antenna turned green and some information about the card was displayed below the picture of the antenna in the simulator. I double-clicked the image of the card next to the antenna nad it “removed” the tag from the android device. I did this with several different tags.

After performing these tests, I saved the trace log file from the trace logger, and I used the android bridge (adb) logcat command to dump the system log from the avd to a file. The first thing I discovered was that the trace logger captured hundreds of thousands of lines of data, but I found those that were specific to my tests. I also isolated the information concerning the open-nfc service that had been logged in the avd’s system log.

The avd system log told me that the open-nfc implementation could process 10 different nfc tag types plus peer-to-peer communications. I then looked throughthe list of virtual tags to find several that were definitely of the specified tag types (ISO14443-4A, ISO 14443-4B, etc). I settled on the micropass\_tag\_card\_4B (ISO14443-3) tag to use for my testcase. The virtual card is implemented as an xml file that looks like this:

<?xml version="1.0"?>

<virtualcard format="1.1">

<cardtype>Micropass</cardtype>

<cardname>[MICROPASS\_TAG\_4B]</cardname>

<image>MICROPASS.jpg</image>

<description />

<detection>

<ISO14443-4>0x05 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x71 0x80 0x00</ISO14443-4>

<ISO14443-3>0x05 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x71 0x80 0x00</ISO14443-3>

</detection>

<data>

<ef id="0xE1 0x03">

0x00 0x0F 0x20 0x00 0xFF 0x00 0xFF 0x04 0x06 0xE1 0x04 0x03 0xFB 0x00 0x00

</ef>

<ef id="0xE1 0x04">

0X00 0X13 0XD1 0X01 0X0F 0X55 0X00 0X77

0X77 0X77 0X2E 0X67 0X6F 0X6F 0X67 0X6C

0X65 0X2E 0X63 0X6F 0X6D

</ef>

</data>

</virtualcard>

The data section of this xml file is what was important. Can I see that either the trace logger or the avd’s log file indicates that this data was read?

The first <ef> section in the tag looks like this:

<ef id="0xE1 0x03">

0x00 0x0F 0x20 0x00 0xFF 0x00 0xFF 0x04 0x06 0xE1 0x04 0x03 0xFB 0x00 0x00

</ef>

In the avd’s log file I find:

I/OPENNFC\_SERVER( 557): 00053 LOG NAL : ND->NFCC NAL-READER\_14\_B-4 CMD-READER\_XCHG\_DATA (len=0x06) 18 00 B0 00 00 0F

I/OPENNFC\_SERVER( 557): 00054 LOG NAL : ND<-NFCC NAL-READER\_14\_B-4 ANS-RES\_OK (len=0x11) 00 0F 20 00 FF 00 FF 04 06 E1 04 03 FB 00 00 90 00

And in the trace log I find:

88053 1 NfcSimulator 09:29.251913 LOG RFDevice1 NFCC<-RF RECEIVING <MSG\_SPECIFIC\_MESSAGE> ; CMD\_XCHG\_DATA\_RESPONSE; SUCCESS; binary[17] 0 f 20 0 ff 0 ff 4 6 e1 4 3 fb 0 0 90 0: 00 0F 20 00 FF 00 FF 04 06 E1 04 03 FB 00 00 90 00

So we can see the flow from the tag through the connectionCenter to the avd for the first piece of card data, but what about the second part of the data?

<ef id="0xE1 0x04">

0X00 0X13 0XD1 0X01 0X0F 0X55 0X00 0X77

0X77 0X77 0X2E 0X67 0X6F 0X6F 0X67 0X6C

0X65 0X2E 0X63 0X6F 0X6D

</ef>In the avd’s log I find:

I/OPENNFC\_SERVER( 557): 00059 LOG NAL : ND->NFCC NAL-READER\_14\_B-4 CMD-READER\_XCHG\_DATA (len=0x06) 18 00 B0 00 02 13

I/OPENNFC\_SERVER( 557): 00060 LOG NAL : ND<-NFCC NAL-READER\_14\_B-4 ANS-RES\_OK (len=0x15) D1 01 0F 55 00 77 77 77 2E 67 6F 6F 67 6C 65 2E 63 6F 6D 90 00

In the trace file we find:

88085 1 NfcSimulator 09:29.296321 LOG RFDevice1 NFCC<-RF RECEIVING <MSG\_SPECIFIC\_MESSAGE> ; CMD\_XCHG\_DATA\_RESPONSE; SUCCESS; binary[21] d1 1 f 55 0 77 77 77 2e 67 6f 6f 67 6c 65 2e 63 6f 6d 90 0: D1 01 0F 55 00 77 77 77 2E 67 6F 6F 67 6C 65 2E 63 6F 6D 90 00

Once again, the data travels from the tag to the ConnectionCenter to the avd.

Obviously the trailing 90 00 is a flag to say this is the end of the message.

In each of these cases, the message just prior to the ones listed in the avd log and the trace file began with the sequence 0x18 0x00 0xb0 which is a request to exchange data the last byte of the strings in green indicates the length of the actual message ot be sent excluding the 0x90 0x00 trailer.

Looking at both the avd log and the trace log, there is no indication that the p2p exchange was ever initiated. This matches what I have read elsewhere that says that the second avd must be either emulating a tag, or have an nfc-enabled app running in the foreground which could “beam” information to the first avd. These avd’s do not come with any apps to either emulate tags or that are nfc-enabled.