SI5/M2II - IoT Security Lab
November 7th, 2022
Yves Roudier - UCA / Polytech Nice Sophia
Smart Card Development Environment Setup

This lab aims at setting up the tools necessary for developing code for Javacards. You will first install the drivers necessary for connecting a *PCSC smart card reader*. You will then install the *Javacard Development Kit* and appropriate *JDK*. Finally, you will install the tools necessary for managing Javacards and uploading Applets and exchanging ISO-7816 APDUs, most notably *gpshell*.

Tools are available on multiple platforms (Linux, Windows, OSX) but we will focus on the Linux platform as it makes it easier to deploy the overall toolchain. You can use your Kali box or another (preferably Debian) Linux distribution.

1) PCSC Smart Card Reader

Contact-based as well as most contactless smartcards can only communicate through an appropriate reader. We will be using PCSC standard drivers to deploy such readers, for it is a widespread and mature technology.

Install the following:

- *apt-get install libusb-dev libusb++-0.1-4c2* (and/or possibly *sudo apt-get install libusb-1.0-0-dev*)
- *apt-get install libccid*
- *apt-get install pcscd*
- make sure a "pcscd" process is running. You can also check the daemon runs with "*sudo service pcscd status*". Otherwise restart the process with "*sudo service pcscd start*"
- *apt-get install libpcsclite1*
- *apt-get install libpcsclite-dev*
- plug in the smartcard reader, then make sure you see the reader's name when typing "*lsusb*"
- *apt-get install libpcsc-perl*
- *apt-get install pcsc-tools*

You can now test the installation with your smartcards and the PCSC tools that have been installed. First of all, run *pcsc_scan*, then insert and remove your smartcards and compare what you see.

Contactless smartcard readers also frequently have a PCSC driver and can be used with similar tools. A number of these contactless cards implement ISO-14443 but not ISO-7816 unless they also have a dual contact based interface.

You can find further references from: http://ludovic.rousseau.free.fr/softwares/pcsc-tools/

2) Javacard Development Kit and JDK

Javacard programs are compiled using the standard javac compiler plus the Javacard Development Kit libraries. You need to install the JDK 1.8 (or lower) for bytecode compatibility reasons. It seems that this JDK is no longer available from Oracle (or is well hidden on Oracle's website) and you will have to install an OpenJDK. You may for instance check the JDK package available at https://cdn.azul.com/zulu/bin/zulu8.66.0.15-ca-jdk8.0.352-linux_amd64.deb. You will find instructions for installation at: https://docs.azul.com/core/zulu-openjdk/install/debian#install-from-azul-apt-repository.

Make sure that "*java -version*" refers to the correct version (1.8 at most).

We will also need to install the Javacard Development Kit or JCDK. The smartcards you will be using rely on Javacard 2.1.1, which is again nowhere to be seen on Oracle's website. You will have to retrieve the library fitting your smartcard from:
https://github.com/martinpaljak/oracle_javacard_sdks
(the whole git can be installed without any later inconvenience).

Oracle provides the documentation for the latest JavaCard (3.1) at the following address (be careful with functionalities unavailable in previous versions of JavaCard):
https://docs.oracle.com/en/java/javacard/3.1/index.html

You will also find unofficial documentations for Javacard 2.1, for instance:
http://aszt.inf.elte.hu/~javabook/java-1.2/javacard/standard/htmldoc/

We will be using real smartcards but please take note that there exist JavaCard simulators that may make it faster and easier to test and debug programs. One is available from Oracle's JCDK itself. The jCardSim environment (https://jcardsim.org) is another possible tool.

The instructions below refer to a direct compilation from the command-line but there are alternatives, for instance automating the compilation with the ant-javacard (https://github.com/martinpaljak/ant-javacard) or IDEs like Eclipse for Javacard (https://eclipse-jcde.sourceforge.net/) or Netbeans (https://cedric.cnam.fr/~bouzefra/cours/ModeEmploi_JC3.0.2.pdf).

You are suggested to configure your environment variables as follows:
export JC_HOME_TOOLS=/home/kali/your-jcdk-git-install/oracle_javacard_sdks-master/jc211_kit
export JAVA_HOME=/usr/lib/jvm/zulu-8-amd64/
export PATH=$JAVA_HOME/bin:$JC_HOME_TOOLS/bin:$PATH

3) Applet Deployment

We also need to use specific tools for managing the card, uploading the applets we compiled through a direct communication with the card. Card Management complies with the Global Platform framework.

This is often managed in card makers or generally proprietary IDEs. If you use Windows, you can try out one such tool called JCIDE (https://javacardos.com/tools?ws=github&prj=jcide). We will be using instead the Global Platform shell also called gpshell, an open source scripted alternative. A more recent tool with similar functionalities called GlobalPlatformPro also

exists but seems incompatible with the GemXpresso Pro cards you will be using and I do not suggest its usage.

The gpshell tool is available from https://kaoh.github.io/globalplatform/ and its documentation can be found at: https://github.com/kaoh/globalplatform/blob/master/gpshell/src/gpshell.1.md . I suggest you install the Homebrew package for gpshell *("brew install kaoh/globalplatform/globalplatform"*) as described in https://github.com/kaoh/homebrew-globalplatform rather than compiling the code. You obviously need to install Brew in the first place (for instance, check and adapt: https://www.how2shout.com/linux/how-to-install-brew-ubuntu-20-04-lts-linux/).

Once gpshell is installed, you can run it as *"gpshell your-script"*. The scripts given in Section 4 should get you started. You can also have a look at other example scripts in /home/linuxbrew/.linuxbrew/share/doc/gpshell2 but many of these are examples for different smartcards with different authentication keys so DON'T RUN THEM UNLESS YOU FULLY UNDERSTAND THE CONSEQUENCES!

Make sure that *gpshell* works by running the following script:
mode_201
enable_trace
establish_context
card_connect
card_disconnect
release_context

which should display the reader name.

4) Communicating with the Smart Card

It is necessary to write some code on the smart card terminal side in order to communicate with one or several applets on the card. There are several alternatives for implementing communications on the terminal-side of an application:

- in early versions of JavaCard, the OpenCard Framework (or OCF – see https://www.openscdp.org/ocf/) provides an abstraction of the card terminal and makes it possible to send APDUs via the card reader. This is currently legacy.
- More recent versions of Javacard (since Javacard 2.2.1) make it possible to use an object-oriented communication through the automated generation of stubs and skeletons.
- JPCSC – the Java PCSC wrapper (https://github.com/klali/jpcsc) provides a simple APDU based interface.
- The javax.smartcardio package (JSR 268) offers a more recent, object-oriented support for PCSC (CardTerminal among a list of CardTerminals, TerminalFactory) making it possible to talk to a Card (isCardPresent(), waitForCardPresent(), connect(), disconnect()) using APDUs (CommandAPDU and ResponseAPDU sent over a CardChannel) – refer to https://docs.oracle.com/javase/7/docs/jre/api/security/smartcardio/spec/javax/smartcardio/package-summary.html

Given the limitations of your Javacards, you will stick to APDU based communication (legacy but portable) and cannot implement the Javacard RMI.

Aside from Java, it is also easy to communicate with the smartcard using APDUs in Python for instance, using the pyscard library (https://pyscard.sourceforge.io/user-guide.html). You might have to install dependencies like swig before doing so (check for instance: https://github.com/LudovicRousseau/pyscard/issues/55).

You can also use the following tools to send APDUs interactively (and generally visualize ATRs)

- gscriptor (installed at Section 1) – see http://ludovic.rousseau.free.fr/softwares/pcsc-tools/
- opensc_tool (installed at Section 1) – see https://github.com/OpenSC/OpenSC/wiki (and more specific documentation is available at: http://htmlpreview.github.io/?https://github.com/OpenSC/OpenSC/blob/master/doc/tools/tools.html)
- PCSC card reader diagnostic tool – see https://www.cardlogix.com/product/pc-sc-smart-card-reader-diagnostic-tool/
- pyApduTool (https://javacardos.com/tools#pyApduTool)

You might also parse your APDUs using the following tools:

- https://smartcard-atr.apdu.fr/ (online parser)
- https://www.javacardos.com/tools/apdu-parser (online parser)
- http://ruimtools.com/atr.php (online parser)
- https://github.com/EIDuy/apdu-parser

Furthermore, you can also check references about APDU responses (beyond the 9000 = OK) at https://eftlab.com/knowledge-base/complet-list-of-apdu-responses.

5) Putting it all together

You can now compile a JavaCard application, called an applet, that will be run on top of the smartcard and upload it to the JavaCard. We will take the example of the HelloWorld applet. Compilation is achieved using

Then the bytecode needs to be verified off-card and wrapped into the .CAP format before being uploaded to the smartcard using the Oracle converter. We also need to supply two identifiers that must not be reserved on the card, namely the applet AID and its package AID. Assuming the HelloWorld class is in package HelloWorld, the compilation should proceed as follows:

javac -source 1.2 -target 1.1 -g -cp /home/kali/SC/oracle_javacard_sdks-master/jc211_kit/bin/api.jar helloWorld/HelloWorld.java

You MUST use source and target otherwise your bytecode will not be compatible.

The .class bytecode file can be converted to a verified class .cap file format as follows:

java -classpath $JC_HOME_TOOLS/bin/converter.jar:. com.sun.javacard.converter.Converter -verbose -exportpath $JC_HOME_TOOLS/api_export_files:helloWorld -classdir . -applet 0xa0:0x0:0x0:0x0:0x0:0x62:0x3:0x1:0xc:0x6:0x1:0x2 HelloWorld helloWorld 0x0a:0x0:0x0:0x0:0x0:0x62:0x3:0x1:0xc:0x6:0x1 1.0

Here, we defined:

- The applet name (helloWorld)
- The applet AID: 0xA00000006203010C060102
- The package AID: 0x0A0000006203010C0601
- You can choose shorter AIDs (according to https://docs.oracle.com/javacard/3.0.5/api/javacard/framework/AID.html, the AID is a sequence of bytes between 5 and 16 bytes in length …)

The *helloWorld.cap* file will be generated in a directory called *javacard* at the same location as the *HelloWorld.class* file.

The applets on the card can be listed using the following *gpshell* script. BE EXTRA CAREFUL NOT TO LOCK THE SMARTCARD (which may happen if you authenticate with the wrong key several times in a row):

```
mode_201
gemXpressoPro
enable_trace
establish_context
card_connect
select -AID A000000018434D
open_sc -security 0 -keyind 0 -keyver 0 -keyDerivation visa2 -key 47454d58505
24553534f53414d504c45  // Open secure channel
get_status -element 40
card_disconnect
release_context
```

In this script, the applet with AID 0xA000000018434D is the Card Manager in charge of uploading and deleting applets. The GemXpresso Pro smartcard makes use of the 0xGEMXPRESSOSAMPLE key described in the script for opening and authenticated secure channel with the Card Manager. This key is the standard key for GemXpresso cards after pre-personalization (you should always look for the key and cryptographic algorithms supported by your Javacard when you buy one).

You can now upload the card with *gpshell*. You need to specify a security domain AID different from that of the Card Manager, as shown in the following *gpshell* script:

```
mode_201
enable_trace
enable_timer
establish_context
card_connect
select -AID A000000018434D00
open_sc -security 3 -keyind 0 -keyver 0 -key 47454d5850524553534f53414d504c45
 -keyDerivation visa2
install -file helloWorld.cap -sdAID A000000018434D00 -nvCodeLimit 4000
card_disconnect
release_context
```

If you want to load a new version of the applet, you need to delete the previous version first, which can be done by deleting the applet and its package using their respective AIDs as outlined in the following gpshell script:

mode_201
gemXpressoPro
enable_trace
enable_timer
establish_context
card_connect
select -AID A000000018434D00
open_sc -security 0 -keyind 0 -keyver 0 -key 47454d5850524553534f53414d504c45
delete -AID a00000006203010c060102
delete -AID 0a0000006203010c0601
card_disconnect
release_context

You might simply go for a unique script deleting the package and applet for every new upload after you uploaded the first version.

Now run again the *gpshell* script listing the applets loaded (see above) on the smartcard and make sure it displays your new AID as selectable.

6) Additional References

Further information can be obtained from the following courses:

- Cartes à Puce, Pascal Urien - https://perso.telecom-paristech.fr/urien/cours_cartes_urien-2008.pdf
- Cours Cartes à Puce, Pascal Chour - https://www.pascalchour.fr/ressources/pccam/cours/cartes.htm
- Cours Secure Smart Objects, Samia Saad Bouzefrane - https://samia.roc.cnam.fr/?page_id=1323
- La carte à microprocesseur – de la carte à mémoire à Java Card – un exemple de système embarqué, Pierre Paradinas - http://cedric.cnam.fr/~paradinas/sem/master/CoursCarte.pdf