A Security Vulnerability of Java Card on Array Access in Financial System

Jiang-pei Xu, Li-ji Wu, Xiang-jun Yang, Yu-zhong Wang, Xiang-min Zhang
Institute of Microelectronics, Tsinghua University
Beijing Smart Card Research Institute of Zhongchao Credit Card Co., Ltd.
Beijing, China
1 xjp11@mails.tsinghua.edu.cn
2 lijiwu@mail.tsinghua.edu.cn
3 yang.xiangjun@gmail.com
4 wang.yuzhong@gmail.com
5 zhxm@mail.tsinghua.edu.cn

Abstract— **Generally, Java Card mainly consists of the following parts: COS (Chip Operating System), JCVM (Java Card Virtual Machine), and API (Application Programming Interface). As a multi-application system, Java Card itself is very complicated, so it may inevitably exist some security vulnerabilities inside. Based on these parts of Java Card, we can find out some detectable points to its security vulnerabilities. This paper presents a method containing a specific case to test Java Card on array access, aiming to detect the possible security vulnerabilities of JCVM. In this paper, three different kinds of Java Cards have been tested and the test result has been described. From the test result, we successfully find out a security vulnerability of JCVM.**

Keywords-Java Card; security vulnerability; array access.

I. INTRODUCTION

Java Card is a smart card that can interpret Java bytecode. Currently, there are two types of Java Card, one uses a CPU for operation, the other one uses both CPU and Java coprocessor. CPU is controlled by software, but the speed of software alone is relatively slow. So there comes the idea of adding a Java coprocessor into Java Card, which not only combines software and hardware together, but also improves the speed of explanation and execution.

The general structure of Java Card is showed as follow:

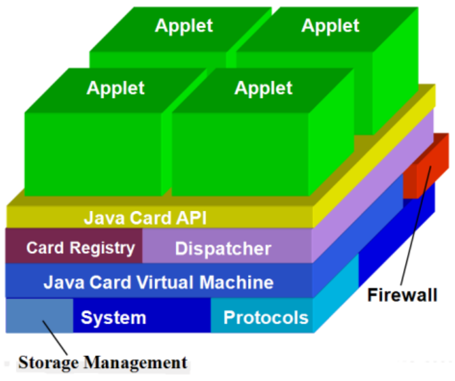


Figure 1. Structure of Java Card

Java Card with multi-application technology is the mainstream of financial IC card. Java Card is widely used in financial IC card because of the following advantages: (i).Cross-platform operation of applets: It is possible to run Java Card applets on Java Cards from different card manufacturers. (ii). Multiple usages: a Java Card can store multiple Java Card applets, different applets are separated by firewalls. (iii).Feature of reuse: A user can remove applets from Java Card or download new applets onto Java Card according to his need. (iv).Compatibility with traditional smart card: Java Card is compatible with the International standard ISO / IEC 7816 and the industry standards (Europay / Master Card / Visa EMV). (v).Simple and rapid development of applets: Developers can develop Java Card applets without necessarily understanding the complex hardware and specialized technology of smart card.

Typical development process of Java Card applets is showed in the figure below:

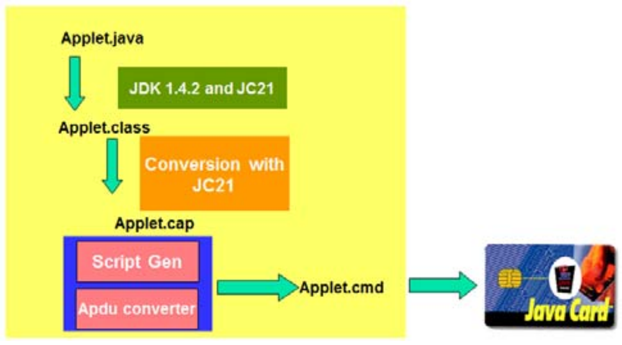


Figure 2. Development process of Java Card

Typical steps to develop Java Card applets are: (i). Write Java source code. (ii). Compile the source code to class file. (iii). Convert the class file into Converted Applet (CAP) file. (iv). Verify if CAP file is valid (this step is optional). (v). Install the CAP file onto Java Card. There is a special and indispensable part of Java Card—Java Card Virtual Machine (JCVM), which is critical for Java Card development. It is a

978-1-4673-5699-2 /13/$31.00 ©2013 IEEE

subset of original Java Virtual Machine and is responsible for literal translation and execution of Java applets. It is the special mechanism of JCVM that makes Java Card have the capability to achieve cross-platform operations. The following block diagram shows the components of JCVM:

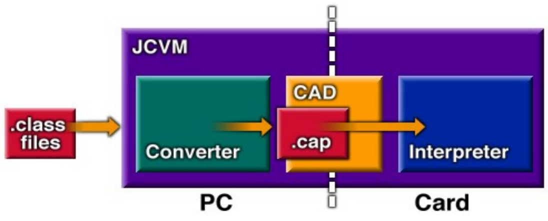


Figure 3. Block diagram of JCVM

As for Java Card, the security of its virtual machine is an important part of the security of embedded software. However, the independent technology of domestic virtual machine is relatively weak, we still have a lot of work to do on its security. And first of all, security problems must be found. Thus it's significant to find out some security vulnerabilities of JCVM and do some test cases on it.

Based on related researchers' previous work, several detectable points to security vulnerabilities of Java Card are already public, which covers the following aspects: fault injections, CAP file manipulation, abusing shareable interface objects, etc. According to their research results, some attacks are effective but some are not, depending on the attack implementation process and the security levels of the tested cards. In this paper, the detectable point of type confusion has been used and an attack case has been created on array access to test three different kinds of Java Cards.

II. METHOD TO TEST JAVA CARD

A. *Overview*

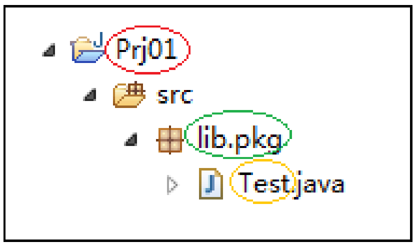
Type checking mechanism of Java Card ensures that no disallowed type conversions are performed. For example, an array can not be converted into an object reference. Technically, type conversions can only be performed by using the "checkcast" instruction, and arguments provided to methods have to be of compatible types.

The method in this paper presents a way to treat a byte array as a short array, it is equivalent to change a "baload" (byte load) opcode into a "saload" (short load). Specific description of the method is presented below.

First step is to create two lib packages. Lib package 1 should be uploaded onto card and lib package 2 is to be used for compilation. Then a Java attack applet should be written. The idea to detect security vulnerability of Java Card on array access is to access a byte array as a short array and see if any data have changed after conversion.

B. *Create lib package 1*

Firstly a Java Card project on JCOP (Java Card Open Platform, a tool for Java Card development) is created as "prj01".Then a package named "lib.pkg" and a class named "Test" are created in "prj01".



Prj01
src
lib.pkg
Test.java

Figure 4. Create lib package 1

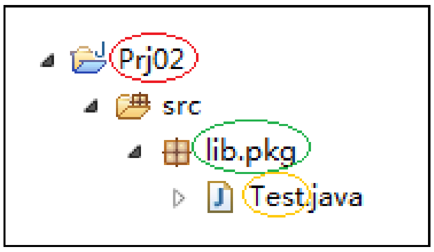
"Test" class in "lib.pkg" of "prj01"mainly consists of the following method:

```
public byte[] ba2sa(byte[] a){
        return a;
        }
```

Note:"ba2sa" is a function name, the parameter and return value are the same byte array named "a".

C. *Create lib package 2*

Secondly another Java Card project is created as "prj02".Then a package named "lib.pkg" and a class named "Test" are created in it.



Prj02
src
lib.pkg
Test.java

Figure 5. Create lib package 2

Note: The AID of lib package 1 and lib package 2 must be the same. Purpose of creating these two similar lib packages is to confuse Java Card. As "compilation" and "upload" are two separated processes, lib package 1 and lib package 2 are used in two different stages.

"Test" class in "lib.pkg" of "prj02" mainly consists of the following method:

```
public short[] ba2sa(byte[] a){
        return null;
        }
```

Note: The return value is null and the type of it is short[].

D. *Create an applet package*

Lastly we create another package named "pkg" in "prj02" (Thus there are two packages in "prj02"). Then create an applet "App01" in "pkg".

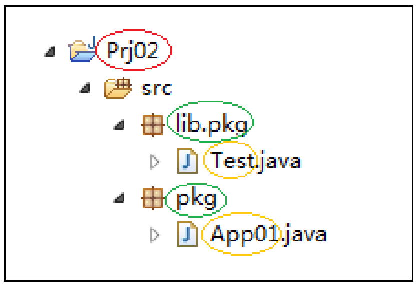


Figure 6. Create an applet package

Note: The applet package and lib package 2 （for compilation）must be in the same project. In applet "App01", lib package 2 is imported as a lib package.

E. Attack Test Applet

The applet "App01" in package "pkg" is the attack test applet. The following figure is the flow diagram of "App01":

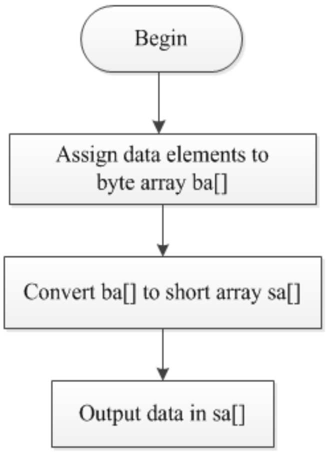

Begin
Assign data elements to byte array ba[]
Convert ba[] to short array sa[]
Output data in sa[]

Figure 7. Flow diagram of "App01"

In "App01", eight data elements are assigned to byte array "ba[]" as follows:

```
private byte ba[] = new byte[] {0x01,0x02,

0x03,0x04,0x05,0x06,0x07,0x08};
```

To access a byte array "ba[]" as a short array "sa[]", we call the method"ba2sa()"as follows:

```
sa = test.ba2sa(ba);
```

In the test applet"App01", lib package 2 is an imported lib package, so after the test applet is created, lib package 2 is used for compilation. When the applet is installed, lib package 1 is uploaded onto card instead.

At last, the access result stored in "sa[]" are output as the test result.

III. RESULT

Three different Java Cards have been tested, all of their versions are v2.2.1.These cards are respectively named as A, B and C, indicating three different manufacturers. Due to the result presented by JCOP, the test result is summarized as follows:

```
A:
  byte array ba[]:
    01   02   03   04   05   06   07   08
  short array sa[]:
  no output, test failed
B:
  byte array ba[]:
    01   02   03   04   05   06   07   08
  short array sa[]:
    0102   0304   0506   0708
    0000   0000   0000   0000
C:
  byte array ba[]:
    01   02   03   04   05   06   07   08
  short array sa[]:
    0102   0304   0506   0708
    0000   000E   0100   020D
```

Figure 8. Test Result

Figure 8 shows that the test applet can not circumvent the security check of Java Card A, attack test on Java Card A is failed. However, the other two cards run the test applet successfully and generate distinguishing results. Java Card B successfully converts eight data elements of byte type to four data elements of short type, and the data in "sa[]" are no more than those in "ba[]". Java Card C also successfully implements the conversion from byte to short, then there are eight data elements of short type in short array "sa[]" , the first four of those are data elements converted from byte to short, but the last four data elements "0000 000E 0100 020D" happen to appear out of expectation, data in "sa[]" are obviously beyond those in "ba[]".

IV. CONCLUTION

Having analyzed the test result, we are able to make the following conclusions: (i) Security policy of Java Card A is stronger than the other two and it has a relatively high security level on array access.(ii)When converting byte array to short array, Java Card B checks array bound in terms of the physical memory size of the array (in bytes), not in terms of the logical size of the array (in number of elements).This check mechanism protects data out of bound from being accessed. (iii) On the contrary, Java Card C checks array bound in terms of the logical size of the array (in number of elements), not in terms of the physical memory size of the array (in bytes), which is not safe because data out of bound can be accessed.

Three tested Java Cards are all smart cards used in financial system. The extra data output by Java Card C is definitely a security vulnerability. No matter what these extra data represent or what they are used for, any unexpected data can threat the security of financial system. So the awareness of users on the security of financial IC card should be strengthened.

REFRENCES

[1] Wojciech Mostowski, Erik Poll. Malicious Code on Java Card Smartcards: Attacks and Countermeasures. Smart Card Research and Advanced Applications, Lecture Notes in Computer Science,vol.5189, 2008, pp.1-16.

[2] Junwei He, Liji Wu, Xiangmin Zhang. Design and Implementation of A Low Power Java Coprocessor for Dual-interface IC Bank Card. The 9th International Conference on ASIC, 2011, pp.1038-1041.

[3] Zeqiri Nderin, Luma Artan. Intelligent communication system, Automation Control, Java Card and Security. Mathematics and Computers in Science and Engineering, 2008, pp.117-121.

[4] Chaumette Serge, Sauveron Damien. An efficient and simple way to test the security of Java Cards. Proceedings of the 3rd International Workshop on Security in Information Systems, WOSIS 2005, in Conjunction with ICEIS 2005, pp.331-341.

[5] Yuchuan Wu, Yaqin Sun. Analysis and research of securing from attack for Java Card. Proceedings of the International Conference on E-Business and E-Government, ICEE 2010, pp.1296-1298.

[6] Wojciech Mostowski. Formalisation and Verification of Java Card Security Properties in Dynamic Logic. The 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005. pp 357-371.

[7] Pieter H.Hartel, Luc Moreau. Formalizing the safety of Java, the Java virtual machine, and Java card. ACM Computing Surveys (CSUR) ,vol.33, 2001, pp.517-558.

[8] Casset L. ,Burdy L., Requet A. Formal development of an embedded verifier for Java Card byte code. Dependable Systems and Networks, 2002, pp.51-56.

[9] Guillaume Bouffard, Julien Iguchi-Cartigny, Jean-Louis Lanet. Combined Software and Hardware Attacks on the Java Card Control Flow. 10th IFIP WG 8.8/11.2 International Conference, CARDIS 2011, pp.283-296.

[10] http://developers.sun.com/techtopics/mobility/javacard/articles/javacard1/