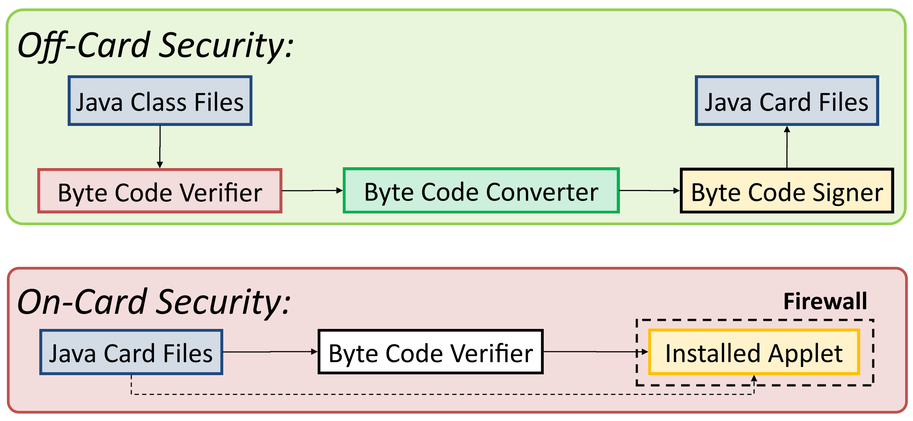
Combined Attacks on the Java Card Smart Cards

Guillaume Bouffard and Jean-Louis Lanet

SSD Team — Xlim/Université de Limoges

{guillaume.bouffard,jean-louis.lanet}xlim.fr

The Java Card Security Model



SSD Tools

- **Opal**

Opal implements the Global Platform Card specification which defines several authentication, encryption and transfer protocols for smart cards.

- **Cap Map**

The Cap Map is a Java 6 library allowing the reading and the modification of Java Card CAP (Converted APplet) files. Thus, you can create and change each component of a CAP file, compatible with the Java Card 3.x Classic Edition specification. Our Java-library returns the (in)valid CAP file.

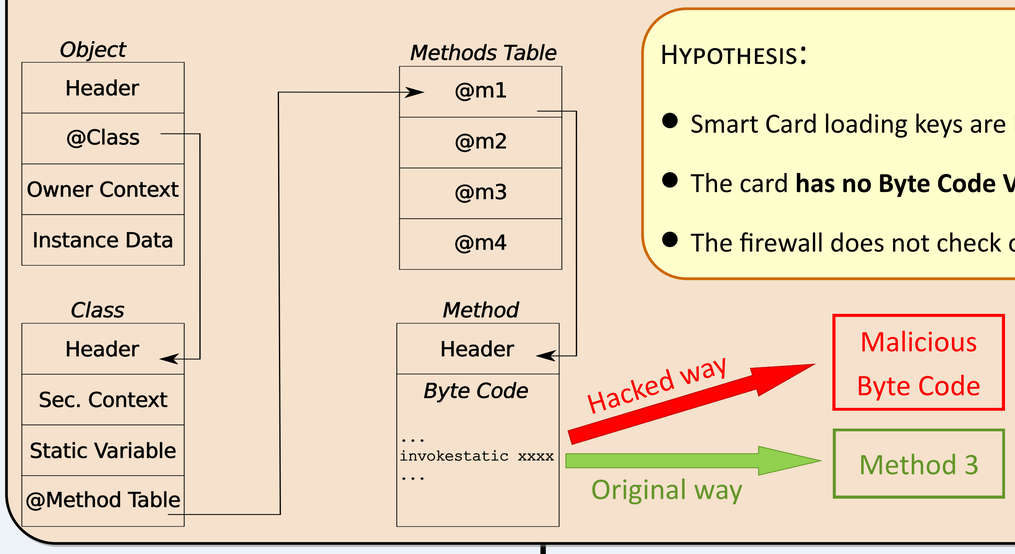
EMAN1: Self-Modifiable Code Generation

Attack aims to abuse Java Card functions on static elements un-checked by the firewall:

- `getstatic`
- `setstatic`
- `invokstatic`

Get Malicious Byte Code Address !

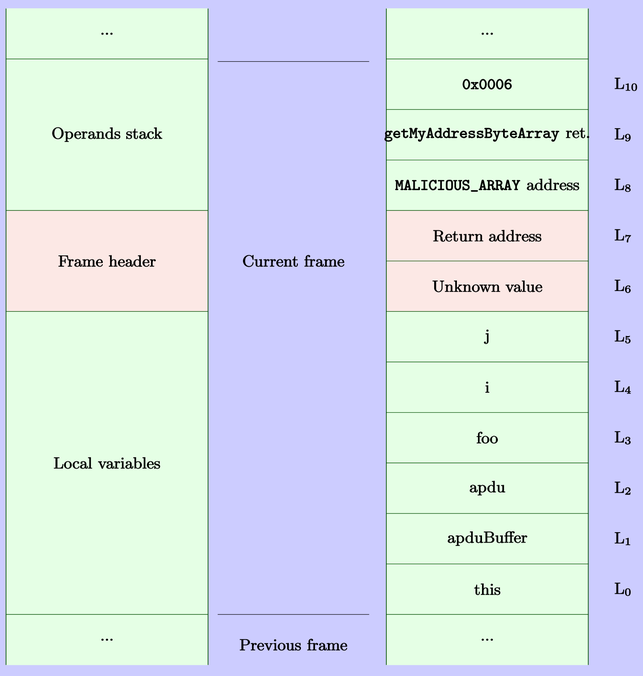
```
public short getNaughtyByteCodeAddress(byte[]bc){
    03 // flags : 0 max_stack : 3
    21 // nargs : 2 max_locals : 1
    10 AA bspush 0xAA
    31    sstore_2
    19    aload_1          <- Push the array address on the stack
    00    nop
    00    nop
    00    nop
    00    nop
    78    sreturn }        <- Return the last pushed short
```



Hypothesis:

- Smart Card loading keys are known
- The card **has no Byte Code Verifier**
- The firewall does not check operations on static

EMAN2: A ghost in the stack




```
public void modifyStack(byte[] apduBuffer,
                        APDU apdu,
                        short foo) {
  short i=(short) 0xCAFE;
  short j=(short)
    getMyAddressByteArray(MALICIOUS_ARRAY)
                  +6);
  i = j; }
```

What is a Mutant?

- Applet is legacy installed
- An external modification change some method byte codes
- This ill-formed applet may execute unauthorized operations.

The Byte Code Verifier Weakness

- The Byte Code Verifier component can be bypassed
- The card must have a hardware or software control flow graph

EMAN4: Mixed Attack

Hypothesis:

- Smart Card loading keys are known
- The card may **have a Byte Code Verifier**

Part of the EEPROM memory map

```
0x0A7F0: 18AE 0188 0018 AE00 8801 18AE 0188 0018
0x0A800: AE00 8801 18AE 0188 0018 AE00 8801 18AE
0x0A810: 0188 0059 0101 A8FF 177A 008A 43C0 6C88
0x0A820: 0000 0000 0000 0000 0000 0000 0000 0000
0x0A830: 0                                     0
0x0A840: 0                                     0
0x0A850: 0                                     0
0x0A860: 0                                     0
0x0A870: 0                                     0
0x0A880: 0                                     0
0x0A890: 0000 0000 0000 0000 0000 0000 0000 0000
0x0A8A0: 0000 0000 0000 1117 128D 6FC0
```

Correct behaviour

```
A8 FF17 goto_w FF17
7A      return
```

Faulty behaviour

```
A8 0017 goto_w 0017
00      nop
...
00      nop
11 1712 sspush 1712
// ISOException.throwIt
8D 6FC0 invokstatic 6FC0
```

Fragment of a method byte code

Our Malicous byte code

Bibliography

Combined Software and Hardware Attacks on the Java Card Control Flow, CARDIS'11, **G. Bouffard, J. Iguchi-Cartigny, J.-L. Lanet**, Leuven Belgium, September 2011

Attacks on Java Card 3.0 Combining Fault and Logical Attacks, Smart Card Research and Advanced Application, **G. Barbu, H. Thiebeauld and V. Guerin**, pp 148--163, Springer 2010

Developing a Trojan applet in a Smart Card, Journal in Computer Virology, **J-L. Lanet, J. Iguchi-Cartigny**, Vol. 6, Issue 4, pp. 343, 2010