

# A Dynamic Syntax Interpretation for Java based Smart Card to Mitigate Logical Attacks

Tiana Razafindralambo, Guillaume Bouffard,  
**Bhagyalekshmy N Thampi**, and Jean-Louis Lanet

*Smart Secure Devices (SSD) Team, XLIM/ Université de Limoges,  
France*

*[bhagyalekshmy.narayanan-thampi@xlim.fr](mailto:bhagyalekshmy.narayanan-thampi@xlim.fr)*

SNDS - 2012

11-12 October 2012



INOSSEM



# Outline

- Introduction
- Java Card Security
  - *Byte code verifier, CAP File, API, Linker, Firewall*
- Types of attacks on Java Cards
- Objective
- Developing a new attack
- Existing countermeasure
- Newly proposed countermeasure & its implementation
- Conclusion

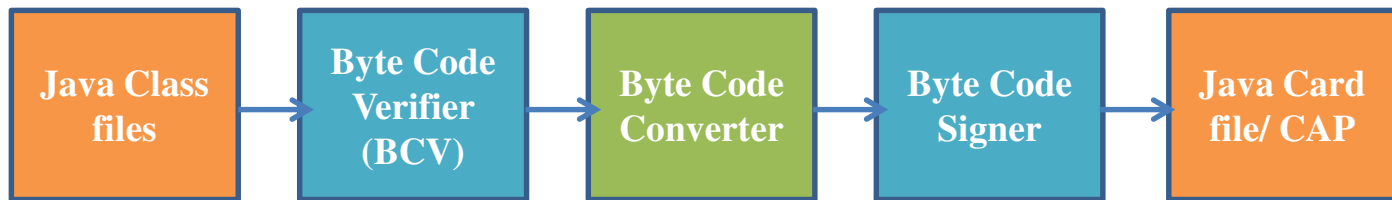
# Introduction

- Smart Card/ Java Card
  - Most of the Smart Cards are Java Card
  - Secure, efficient, cost effective embedded device
  - Limited memory size (RAM, ROM, EEPROM)
  - Prone to attacks
  - Hardware & software security
  - Multi-application environment

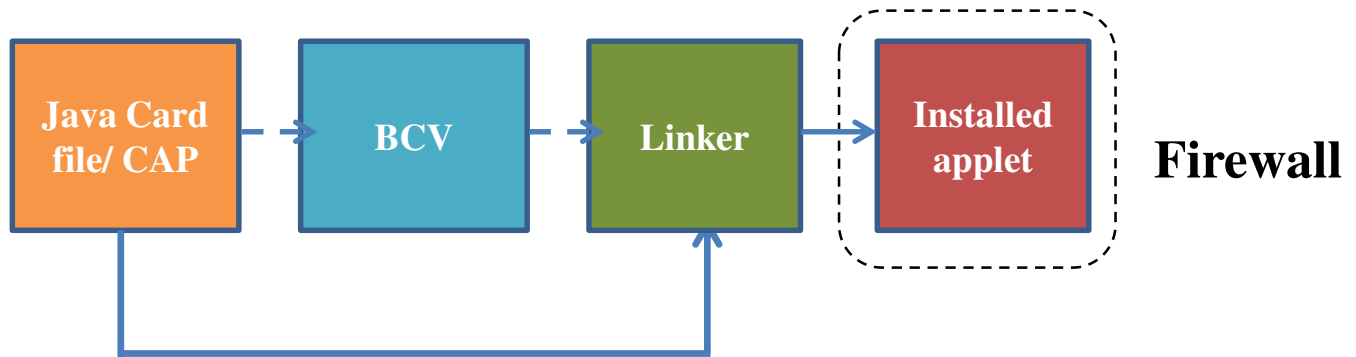


# Java Card Security

## Off-Card Security Model



## On-Card Security Model



# Java Card Security: CAP File

- CAP: Converted Applet
- Binary representation of a package of classes
- Consists of 12 components
- Some of the main components
  - Class
  - Method
  - Constant Pool
  - Reference Location etc.

# Types of attacks on Smart Card

- **Logical**
  - software/ sensitive informations
  - two categories of logical attacks
    - well formed CAP File: shareable interface mechanism, transaction mechanism
    - ill formed CAP File: CAP File manipulation
- **Side Channel**
  - cryptographic secrets obtained through electromagnetic leaks, timing information, power consumption, heat radiation, etc.

# Types of attacks on Smart Card (Contd.)

- **Physical**
  - fault attacks (optical, electromagnetic)
  - input current modifications
  
- **Combined**
  - logical and physical
    - fault injection (bypass on-card BCV)

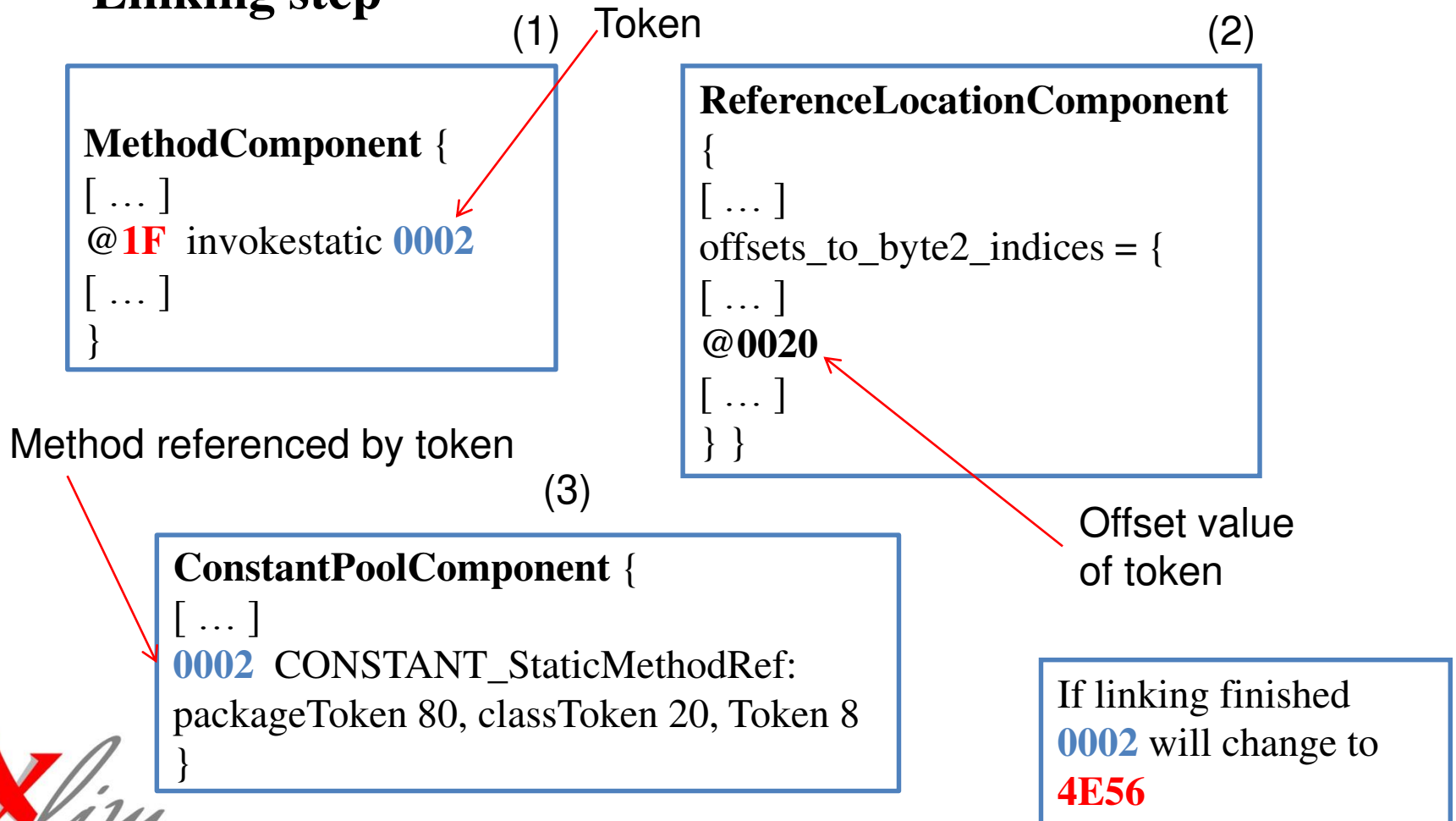
# Objective: introducing a new logical attack

- Abused the Java Card linker to change the correct bytecode into malicious one
- Set of instructions modified
- Each instruction is referenced by an offset in the method component
- Linking step is done during the loading of a CAP file
- Linker interprets the instructions as tokens and resolve it
- CAP File Manipulator: developed by our team
  - Allows to read and modify Cap Files or any component of a CAP File
  - Respect the interdependencies between the components



# Objective: introducing a new logical attack (Contd.)

## Linking step



# Developing a new attack

## Method Component

Offset	Bytecode	Mnemonic
0020	[0x00]	nop
0021	[0x02]	sconst_m1
0022	[0x02]	sconst_m1
0023	[0x3C]	pop2
0024	[0x04]	sconst_m1
0025	[0x3B]	pop

Linking needs  
two bytes

Constant Pool Component  
/\* 0008, 2 \*/CONSTANT\_StaticMethodRef:  
external: 0x80, 0x8, 0xD

Reference Location Component  
Offset value: 0020

# Developing a new attack (Contd.)

## Set of instructions after linking resolution

Offset	Bytecode	Mnemonic
0020	[0x8E]	Invokeinterface
0021	[0x03]	<i>// nargs</i>
0022	[0x02]	<i>// indexByte1</i>
0023	[0x3C]	<i>// indexByte2</i>
0024	[0x04]	<i>// method</i>
0025	[0x3B]	pop

Token method **0x0002** is linked by the value **0x8E03**

# Existing countermeasure

$$ins_{hidden} = ins \oplus K_{bytecode} \quad (1)$$

where  $K_{bytecode}$  is the key,  $ins$  is the instruction

- Impossible to execute the malicious code without the knowledge of  $K_{bytecode}$
- To find xor key: change the Control Flow Graph (CFG)
- Through brute force attack: easily obtain xor key with 256 possible values

# Newly proposed countermeasure

$$ins_{hidden} = ins \oplus K_{bytecode} \quad (1)$$

$$ins_{hidden} = ins \oplus K_{bytecode} \oplus jpc \quad (2)$$

*Scrambling Bytecode with*

*equation 1*

*equation 2*

Address	Bytecode	Mnemonic
0x8068	0x42	nop
0x8069	0x40	sconst_m1
0x806A	0x40	sconst_m1
0x806B	0x7E	pop2
0x806C	0x46	sconst_1
0x806D	0x79	pop

Address	Bytecode	Mnemonic
0x8068	0x2a	nop
0x8069	0x29	sconst_m1
0x806A	0x2a	sconst_m1
0x806B	0x15	pop2
0x806C	0x2d	sconst_1
0x806D	0x12	pop

# Countermeasure implementation (Contd.)

## Unscrambling shell code

Offset	Bytecode	Mnemonic
0xAB80	0x7D	getstatic 8000
0xAB83	0x78	sreturn

## After unmasking each instruction

Offset	Bytecode	Mnemonic
0xAB80	0xBF	<i>//undefined</i>
0xAB81	0x43	ssub
0xAB82	0xC0	<i>// undefined</i>
0xAB83	0xB9	<i>// undefined</i>

# Conclusion

- Based on the vulnerability of the linker, a powerful logical attack demonstrated
  - *Correct bytecode to into malicious one*
- Protect Java Card from logical attacks
  - *Impossible to execute malicious bytecode without the knowledge of jpc stored in the EEPROM*
- Cost effective countermeasure, suitable for security interoperability

# Future Work

- To do reverse engineering using electromagnetic side channel attacks



# THANK YOU



Bhagyalekshmy N THAMPI, Research Engineer  
bhagyalekshmy.narayanan-thampi@xlim.fr  
Smart Secure Devices (SSD) Team  
XLIM/ Université de Limoges, 123 Avenue Albert Thomas, 87060 Limoges,  
France  
<http://secinfo.msi.unilim.fr/>