A friendly framework for hiding fault enabled virus for Java based smart cards

Tiana Razafindralambo Guillaume Bouffard
Jean-Louis Lanet

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

DBSec'12

Smart Card

Smart card is...

- Tamper-Resistant Computer
- Securely stores and processes information
- Used in our everyday life:
  - SIM card
  - Credit Card
  - Health Insurance Card, etc.
- Most of the smart cards are based on Java Card technology

It contains critical information!

Objectives

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

Context

Smart card attacks:

- Physical:
  - Voltage modification (execution flow)
  - Light injections to the memory cells (LED, laser, etc.)
  - Side channel attacks (power and electromagnetic analysis, etc.)
- Logical:
  - CAP file manipulation (bypass off-card BCV)
  - shareable interface mechanism (no longer possible)
  - transaction mechanism
- Combined attack:
  - fault injection (bypass on-card BCV)

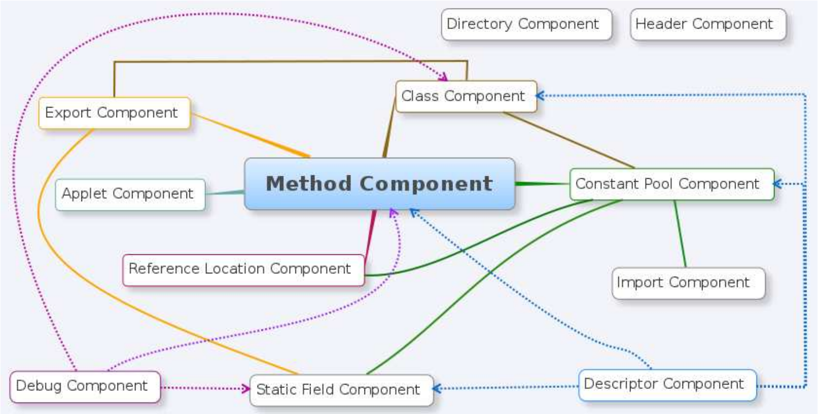
Context

Two categories of logical attacks:

- Ill-typed application: modification of input file **(1)**
- Well-typed application:
  - specification weakness
  - application mutation **(1)**

(1) Byte code transformation engineering at CAP file level

Converted APplet (CAP) File



The CAP File Manipulator

CAP MAP

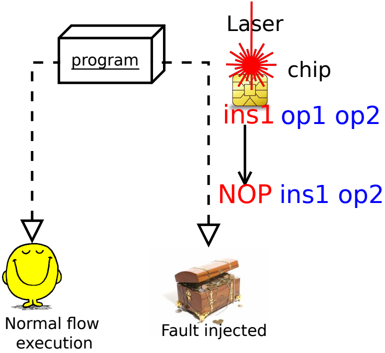
- Java-based framework
- Allows to read any element of a CAP file
- Modification of any component of a CAP file
- Friendly tool to design logical attacks

CAP MAP
Cap File Manipulator

Fault enabled viruses

Our mission

Build a program that may have two semantics execution



Example : get the secret key

```
public void process(APDU apdu) {
    short localS; byte localB;
    // get the APDU buffer
    byte[] apduBuffer = apdu.getBuffer();
    if (selectingApplet()) { return; }
    byte receivedByte = (byte) apdu.
        setIncomingAndReceive();
                                                B1

    // any code can be placed here
    // ...

    DES_keys.getKey(apduBuffer, (short)0);     B2

    apdu.setOutgoingAndSend((short)0,16) ;     B3

}
```

Constraints solving

Domain definition

$\mathbb{I} = \{\mathbb{I}_0, ..., \mathbb{I}_n\}$
$\mathbb{I}_0 = \{\{ins_1, ins_2, ...\}/operands = 0\}$
$\mathbb{I}_n : \{\{ins_1, ins_2, ...\}/operands = n\}$
$\mathbb{S} : Stack$

Constraints solving

Constraints:

Find $\mathbb{X}_1 = \{ins_1, ins_2, ...\}$ such as:

- $\mathbb{X}_1 \in \mathbb{I}_n$
- $|\mathbb{S}_1| > |\mathbb{S}_0|$
- $|\mathbb{S}_i| \leq MaxStack$
- number of locals must not change
- $\forall ins_i \in \mathbb{X}_1, \mathbb{S}_0(ins_i) = \mathbb{S}_1(ins_i)$

Address resolution

```
// any code can be placed here
//...
DES_keys.getKey(apduBuffer, (short)0);
```

Listing 1: Original B2 to hide

Address resolution

```
OFFSETS            INSTRUCTIONS          OPERANDS

...
/*00d4*/           nop
/*00d5*/           nop
/*00d6*/           getfield_a_this  1   // DES_keys
/*00d8*/           aload            4   // L4=>apdubuffer
/*00da*/           sconst_0
/*00db*/           invokeinterface  nargs: 3, index: 0,            B2
                                    const: 3, method: 4
/*00e0*/           pop                  // return byte
```

Address resolution

```
OFFSETS        INSTRUCTIONS        OPERANDS

...
/*00d4*/       nop
/*00d5*/       nop
/*00d6*/       getfield_a_this     1   // DES_keys
/*00d8*/       aload               4   // L4=>apdubuffer
/*00da*/       sconst_0
/*00db*/       invokeinterface     nargs: 3, index: 2,        B2
                                   const: 60, method: 4
/*00e0*/       pop                     // return byte
```

Address resolution

OFFSETS	INSTRUCTIONS	OPERANDS	
...			
/*00d4*/	nop		
/*00d5*/	nop		
/*00d6*/	getfield_a_this	1 // DES_keys	
/*00d8*/	aload	4 // L4=>apdubuffer	
/*00da*/	sconst_0		
/*00db*/	invokeinterface	03, 02, 3C, 04	B2
/*00e0*/	pop	// return byte	

Hide your code

```
OFFSETS        INSTRUCTIONS      OPERANDS

...
/*00d4*/       nop
/*00d5*/       nop
/*00d6*/       getfield_a_this   1   // DES_keys
/*00d8*/       aload             4   // L4=>apdubuffer
/*00da*/       sconst_0
/*00db*/       invokeinterface   03, 02, 3C, 04          B2
/*00e0*/       pop                  // return byte
```

Hide your code

```
OFFSETS         INSTRUCTIONS        OPERANDS

...
/*00d4*/        nop
/*00d5*/        getfield_a_this     1    // DES_keys
/*00d6*/        aload               4    // L4=>apdubuffer
/*00d7*/        sconst_0
/*00d8*/        ifle                ??
/*00d9*/        invokeinterface     03, 02, 3C, 04          B2
/*00de*/        pop                      // return byte
```

Hide your code

```
OFFSETS        INSTRUCTIONS       OPERANDS

...
/*00d4*/       nop
/*00d5*/       getfield_a_this    1    // DES_keys
/*00d6*/       aload              4    // L4=>apdubuffer
/*00d7*/       sconst_0
/*00d8*/       ifle               8E
/*00da*/       sconst_0                                    B2
/*00db*/       sconst_m1
/*00dc*/       pop2
/*00de*/       sconst_1
/*00df*/       pop                     // return byte
```

Code mutation

```
OFFSETS          INSTRUCTIONS         OPERANDS

...
/*00d4*/         nop
/*00d5*/         getfield_a_this  1   // DES_keys
/*00d6*/         aload            4   // L4=>apdubuffer
/*00d7*/         sconst_0
/*00d8*/         ifle                 8E
/*00da*/         sconst_0                                  B2
/*00db*/         sconst_m1
/*00dc*/         pop2
/*00de*/         sconst_1
/*00df*/         pop                 // return byte
```

Code mutation

```
OFFSETS          INSTRUCTIONS         OPERANDS

...
/*00d4*/         nop
/*00d5*/         getfield_a_this      1   // DES_keys
/*00d6*/         aload                4   // L4=>apdubuffer
/*00d7*/         sconst_0
/*00d8*/         ??                   8E
/*00da*/         sconst_0                                       B2
/*00db*/         sconst_m1
/*00dc*/         pop2
/*00de*/         sconst_1
/*00df*/         pop                      // return byte
```

Code mutation

```
OFFSETS          INSTRUCTIONS        OPERANDS

...
/*00d4*/         nop
/*00d5*/         getfield_a_this    1    // DES_keys
/*00d6*/         aload              4    // L4=>apdubuffer
/*00d7*/         sconst_0
/*00d8*/         00                 8E
/*00da*/         sconst_0                                      B2
/*00db*/         sconst_m1
/*00dc*/         pop2
/*00de*/         sconst_1
/*00df*/         pop                     // return byte
```

Code mutation

```
OFFSETS          INSTRUCTIONS        OPERANDS

...
/*00d4*/         nop
/*00d5*/         getfield_a_this     1    // DES_keys
/*00d6*/         aload               4    // L4=>apdubuffer
/*00d7*/         sconst_0
/*00d8*/         nop                 8E
/*00da*/         sconst_0                                 B2
/*00db*/         sconst_m1
/*00dc*/         pop2
/*00de*/         sconst_1
/*00df*/         pop                      // return byte
```

Code mutation

```
OFFSETS          INSTRUCTIONS          OPERANDS

...
/*00d4*/         nop
/*00d5*/         getfield_a_this       1    // DES_keys
/*00d6*/         aload                 4    // L4=>apdubuffer
/*00d7*/         sconst_0
/*00d8*/         nop
/*00d9*/         invokeinterface       03, 02, 3C, 04          B2
/*00de*/         pop                        // return byte
```

1. Introduction
   - Smart Card
   - Objectives
   - Context
   - CAP File Manipulation
2. Fault enabled viruses
   - Example: get the secret key
   - Constraints solving
   - Address resolution
   - Hide your code
   - Code mutation
3. Conclusion

Conclusion

- Contributions: basic constraints solver, stack evaluator
- CAP MAP is a friendly framework that gives us the ability to:
  - make CAP file manipulation in a coherent way
  - design logical attacks
  - design fault enabled viruses
- SmartCM, a static analyzer to detect fault enabled viruses
- Our future works includes: second order viruses

Thank you for your attention !


```
> aina.razafindralambo@etu.unilim.fr
> guillaume.bouffard@xlim.fr
> jean-louis.lanet@xlim.fr
> http://secinfo.msi.unilim.fr
```