A Secure Virtual Machine For Java Card Platform

Ahmadou A. Séré, Julien Iguchi-Cartigny, Jean-Louis Lanet

SSD Team

Outline

Introduction

Countermeasures

Evaluation

Conclusion

Java Card

Java implementation

- Subset of Java
- Targets embedded systems with limited capacities

Two versions of Java Card

- Java Card 3 classic edition
- Java Card 3 connected edition

Fault Attacks

Physical attack

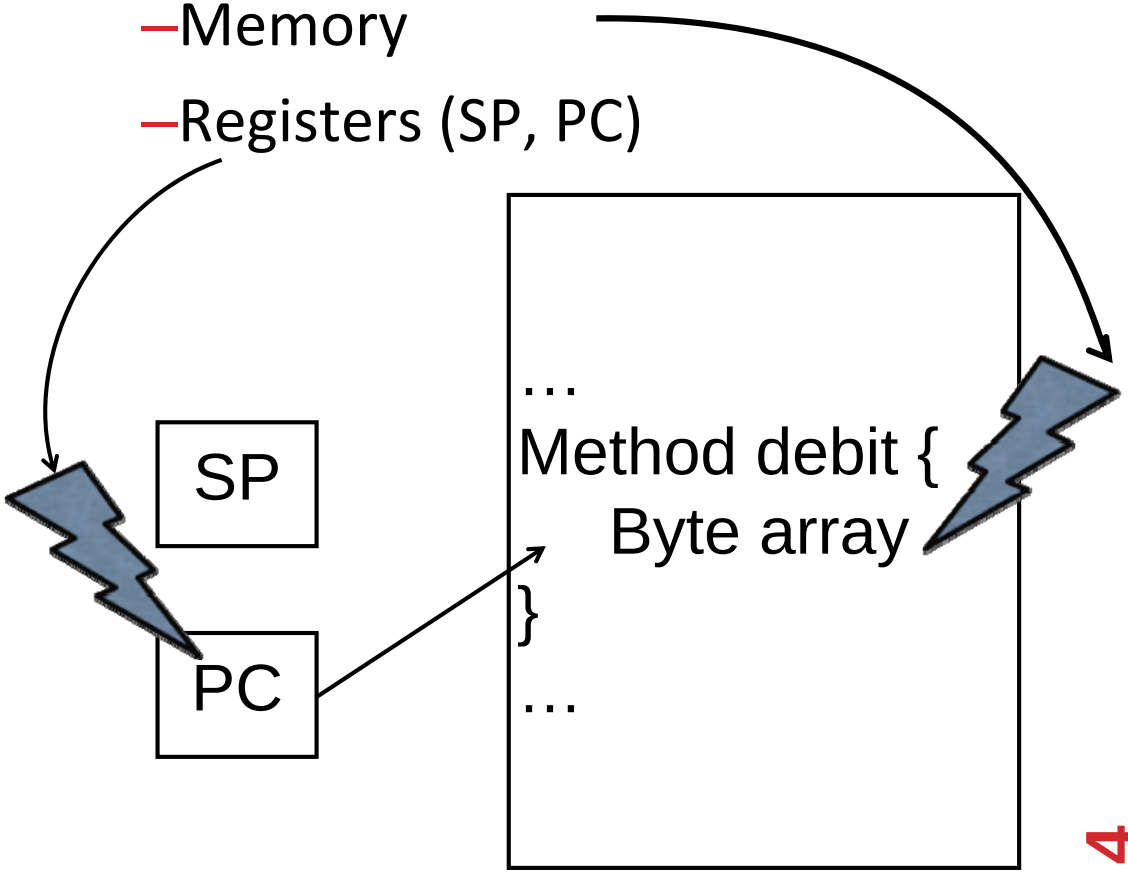
- Electromagnetic attacks (EMA)
- Laser

Two categories

- Invasive
- Non Invasive

Disturb component

- Memory
- Registers (SP, PC)



Fault Model

- **4 criteria**
  - Location
  - Timing
  - Precision
  - Fault type

Fault Model

Fault model	Timing	precision	location	fault type	Difficulty
Precise bit error	total control	bit	total control	set (1) or reset (0)	++
Precise byte error	total control	byte	total control	set (0XFF), reset (0x00) or random	+
Unknown byte error	loose control	byte	no control	set (0XFF), reset (0x00) or random	-
Unknown error	no control	variable	no control	set (0XFF), reset (0x00) or random	--

Mutant Application

- **Java Card Application**
  - Modified by an attack
  - Have sense for the virtual machine interpreter
  - Potentially dangerous for the smart card security

Mutant application

Bytecode

```
00 : aload_0
01 : getfield #4
04 : invokevirtual #18
07 : ifeq 59
10 :  …
…
59 : sipush 25345
63 : invokestatic #13
66 : return
```

Octets

```
00 : 18
01 : 83 00 04
04 : 8B 00 23
07 : 60 00 3B
10 : …
…
59 : 13 63 01
63 : 8D 00 0D
66 : 7A
```

Java code

```
private void debit(APDU apdu) {


  if ( pin.isValidated() ) {
     // make the debit operation
  } else {
     ISOException.throwIt (
       SW_PIN_VERIFICATION_REQUIRED);
}
```

Bytecode

```
00 : aload_0
01 : getfield #4
04 : invokevirtual #18
07 : nop
08 : nop
09 : pop
10 :  …
…
59 : sipush 25345
63 : invokestatic #13
66 : return
```

Octets

```
00 : 18
01 : 83 00 04
04 : 8B 00 23
07 : 00
08 : 00
09 : 3B
10 : …
…
59 : 13 63 01
63 : 8D 00 0D
66 : 7A
```

Java code

```
private void debit(APDU apdu) {

  if ( pin.isValidated() ) {
     // make the debit operation

  } else {
    ISOException.throwIt (
    SW_PIN_VERIFICATION_REQUIRED);
  }
```

A Secure VM

Compliant with the Sun specification

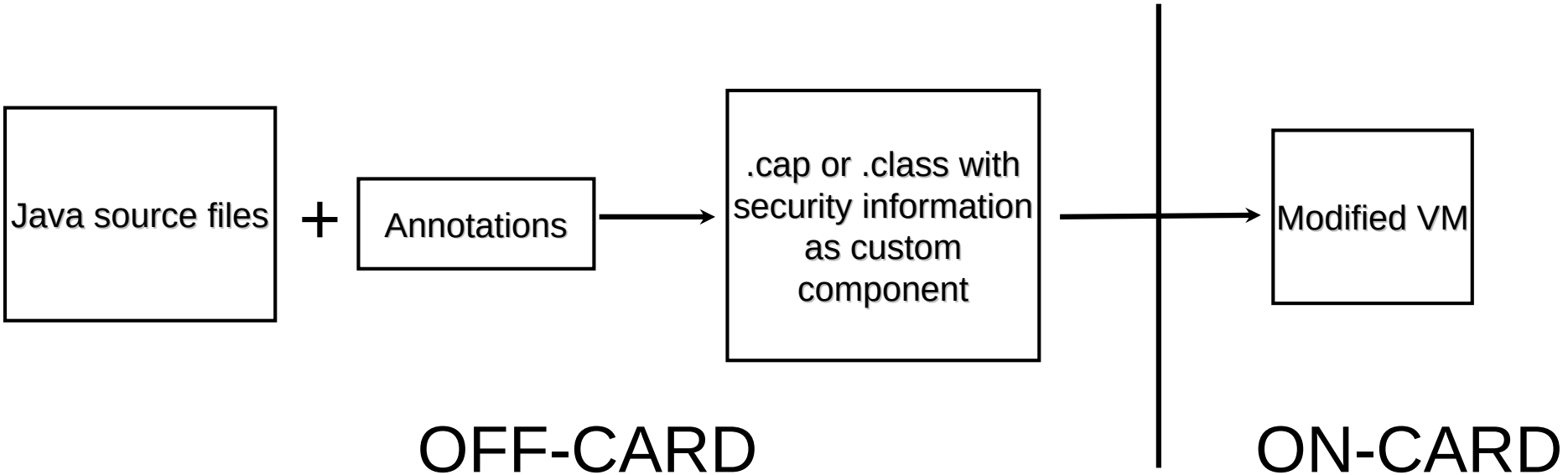
- Run application safely
- Detect code modification

Detect control flow modification

Resources constraints

- Memory consumption
- CPU overhead

Approach




```
@SensitiveType{
 sensitivity= SensitiveValue.INTEGRITY,
 proprietaryValue="PathCheck"
}
private void debit(APDU apdu) {
if ( pin.isValidated() ) {
//make the debit operation
} else {
ISOException.throwIt (SW_PIN_VERIFICATION_REQUIRED);
}
```

Path check mechanism

Goal:

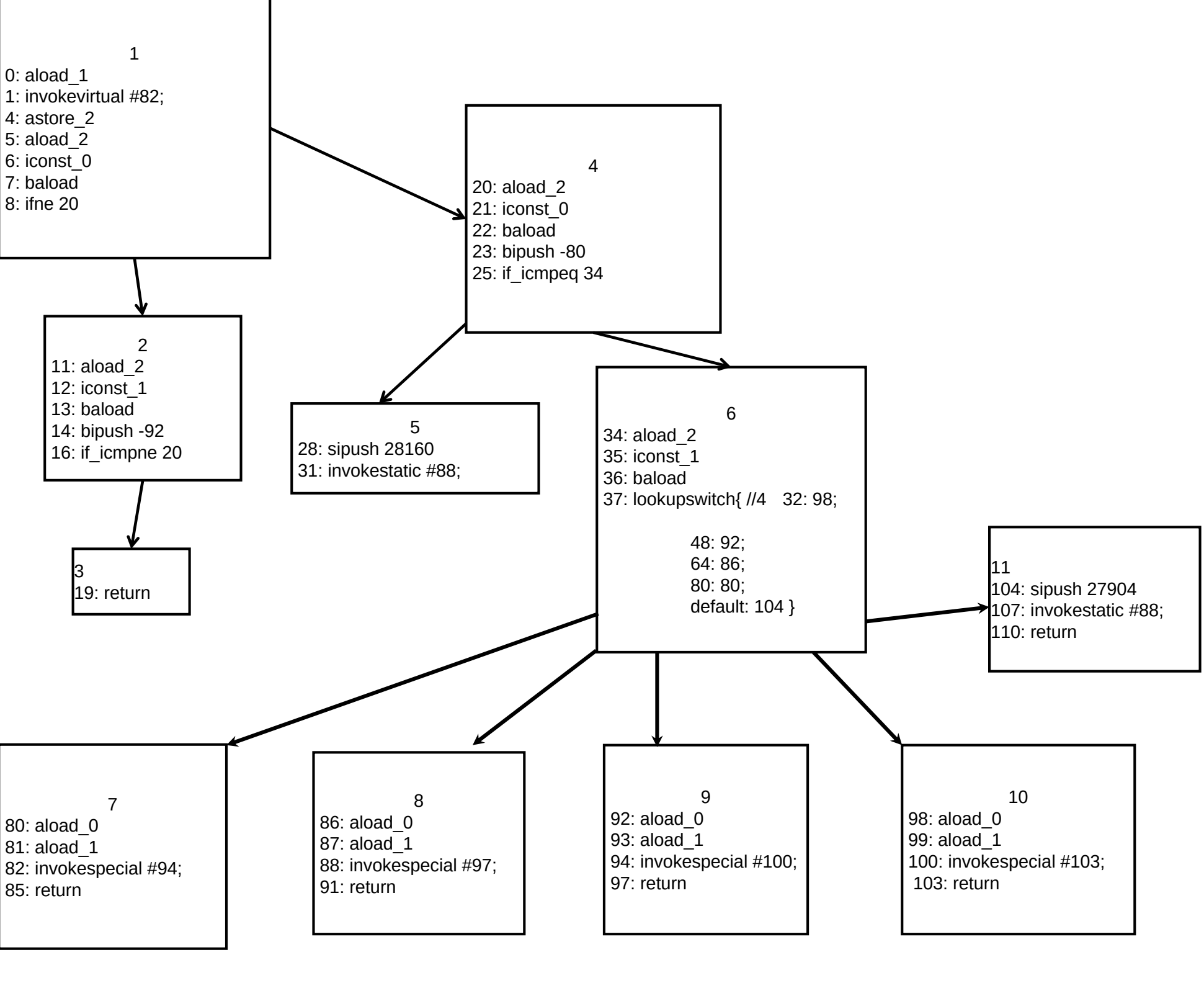
- Detect PC modification during control flow transfer

Principle

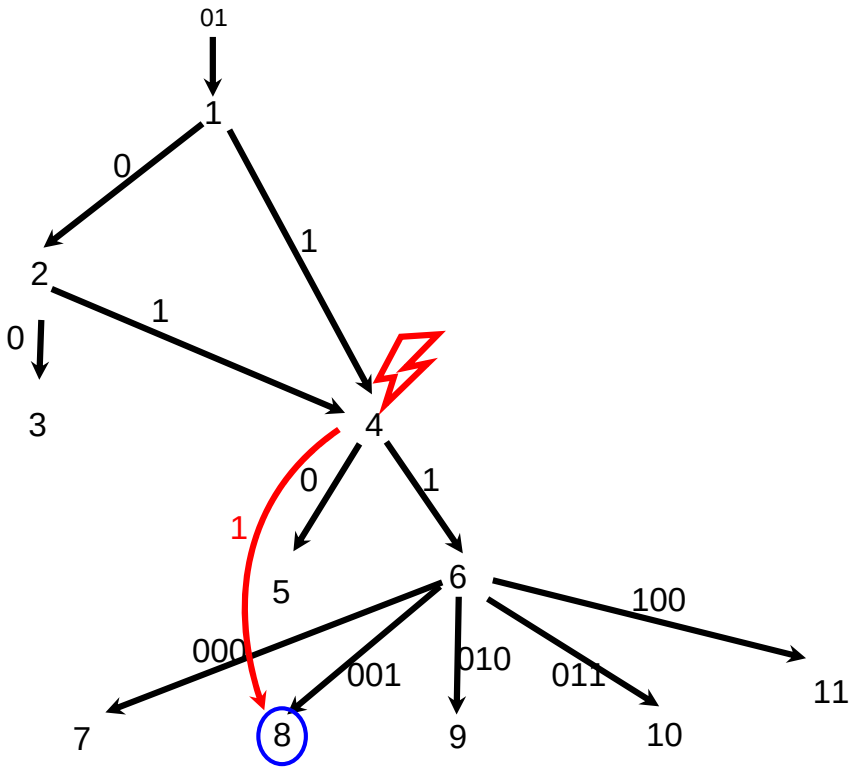
- Off-card
  - Create a Control Flow Graph (CFG)
  - Find all potential paths in this graph and memorize them
- On-card
  - Compute path during runtime
  - Check path concordance between the previously saved one

Path check mechanism

```
public void process(javacard.framework.APDU);
Code:   Stack=2, Locals=3, Args_size=2
 0: aload_1
 1: invokevirtual #82; //Method javacard/framework/APDU.getBuffer:()[B
4: astore_2
5: aload_2
6: iconst_0
7: baload
8: ifne        20
11: aload_2
12: iconst_1
13: baload
14: bipush -92
16: if_icmpne          20
19: return
20: aload_2
21: iconst_0
22: baload
23: bipush -80
25: if_icmpeq          34
28: sipush 28160
31: invokestatic #88; //Method
javacard/framework/ISOException.throwIt:(S)V
34: aload_2
35: iconst_1
36: baload
37: lookupswitch{ //4
              32: 98;
              48: 92;
              64: 86;
              80: 80;
              default: 104 }
80: aload_0
81: aload_1
82: invokespecial #94; //Method getBalance:(Ljavacard/framework/APDU;)V
85: return
86: aload_0
87: aload_1
88: invokespecial #97; //Method debit:(Ljavacard/framework/APDU;)V
91: return
92: aload_0
93: aload_1
94: invokespecial #100; //Method credit:(Ljavacard/framework/APDU;)V
97: return
98: aload_0
99: aload_1
100: invokespecial #103; //Method verify:(Ljavacard/framework/APDU;)V
103: return
104: sipush27904
107: invokestatic #88; //Method
javacard/framework/ISOException.throwIt:(S)V
110: return
```


1
0: aload_1
1: invokevirtual #82;
4: astore_2
5: aload_2
6: iconst_0
7: baload
8: ifne 20
2
11: aload_2
12: iconst_1
13: baload
14: bipush -92
16: if_icmpne 20
3
19: return
4
20: aload_2
21: iconst_0
22: baload
23: bipush -80
25: if_icmpeq 34
5
28: sipush 28160
31: invokestatic #88;
6
34: aload_2
35: iconst_1
36: baload
37: lookupswitch{ //4 32: 98;
48: 92;
64: 86;
80: 80;
default: 104 }
11
104: sipush 27904
107: invokestatic #88;
110: return
7
80: aload_0
81: aload_1
82: invokespecial #94;
85: return
8
86: aload_0
87: aload_1
88: invokespecial #97;
91: return
9
92: aload_0
93: aload_1
94: invokespecial #100;
97: return
10
98: aload_0
99: aload_1
100: invokespecial #103;
103: return

Path check mechanism



CFG is represented by
- A list of vertices
- A list of edges

Edges retain principally three information:
edges {Orig, Dest, Code}

To find the node that has the number 8:

1) ch: 1 2 4 6 8 ⇔ 01 0 1 1 001

2) ch: 1 4 6 8 ⇔ 01 1 1 001

1)
ch 01 0 1 1 001
ch' 01 0

2)
ch 01 1 1 001
ch' 01 1 1

Evaluation Platform

At91 EB40A evaluation board

- Arm7 CPU
- Same resources as regular smart card

Simple RTJ

- Tiny Java Virtual machine
- Small memory and limited CPU capacities

Abstract Interpreter (AI)

Simulate method frame

- Stack
- Local variable table

Simulate PC move

Implement parts of Byte Code verifier during runtime (VF1)

Simulate fault attack

Mutant Generator

Build on top of AI

Generate all the mutant of a given application

- Attack on all the pc of a method
- Generate mutant application when no detection is made

Resources usage

Overhead	CPU	EEPROM	ROM
Path check	< 8%	Variable (0 to 7%)	≈ 1%

Mutant reduction

	No verifications	Partial BCV Checks	Path Check
Number of mutant	440	54	37
Latency (AVG)	-	1,51	2,52
Execution time	7 s	5 s	2 s

Wallet non encrypted memory - 636 attacks - 236 instructions

Conclusion

Countermeasure proposed

- Affordable for the card
- Respectful of Java Card specification
- Don’t need for a developer to be aware of the underlying security mechanisms
- Need small changes on VM interpreter
- Portability of application

Resources

http://secinfo.msi.unilim.fr

http://msi.unilim.fr/~sere

ahmadou-al-khary.sere@xlim.fr

ANY QUESTIONS ?

Thank you for your attention