Evaluation of the Ability to Transform SIM Application into hostile Applications

Guillaume Bouffard, Jean-Louis Lanet, Jean-Baptiste Malchemie, Yves Poichotte and Jean-Philippe Wary

SSD Team – Xlim/Université de Limoges
SFR, Group Fraud & Information Security Direction

guillaume.bouffard@xlim.fr

Outline

I. SFR Presentation
II. What is a Mutant application?
III. The Fault Model
IV. Counter-Measure: Path-Check
V. SmartCM
VI. Metrics
VII. Conclusion

SFR Presentation

SFR, 1st alternative operator on all telecoms market segments

SFR covert all segments of the French telecom market

Consumer

Enterprise

Wholesale

SFR addresses 1 french out of 2

21.3m mobile customers

150K enterprise customers

4.9m broadband Internet customers

200 Operators and 10 MVNOs

Leading Mobile Broadband network

- 18 000 radio sites
- 99% 2G coverage
- 94% 3TG coverage

The 1rst alternative Fixed Broadband infrastructure

- 76% unbundled ADSL coverage
- 57 000 km fiber backbone
- 3m Wifi hotspots

Group Fraud & Information Security

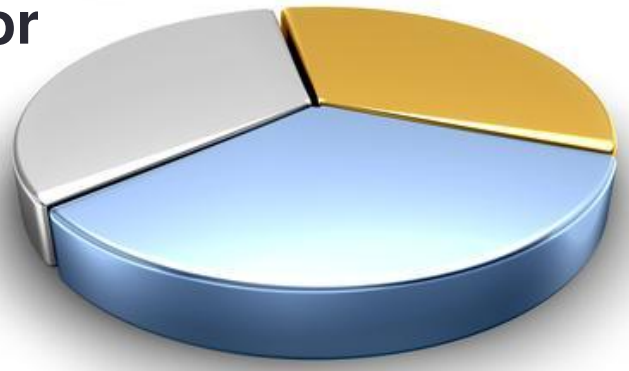
Missions

- Security Expertise: Security recommendation for operational and Business Units
- Anticipation & Intelligence: business intelligence, security and anti fraud knowledge as added value services
- Governance: Fraud & Security risk management

Main Objectives

Trusted Operator

- Neutrality Approach
- Privacy Protection
- Legal Compliance



Value-Added Services

- Processes Industrialization
- Innovative Methodology
- Business-oriented

Business Enabler

- Business Intelligence
- Contextual Security
- Proof of Concept

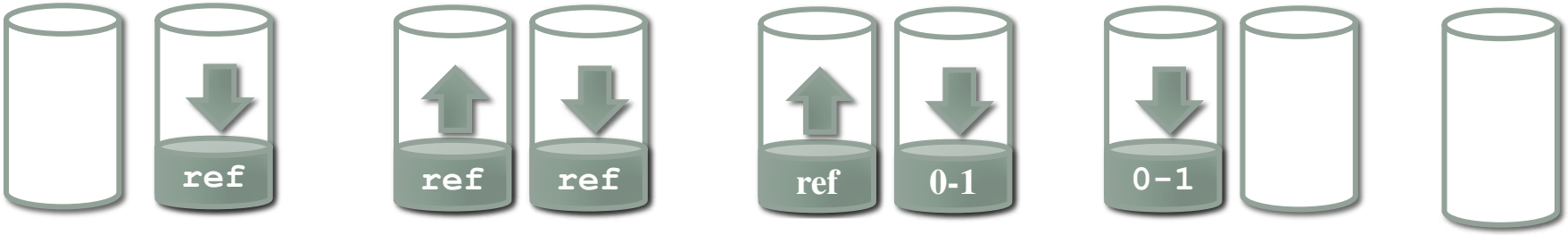
Mutant

- Definition
  - A piece of code that passed the BC verification during the loading phase or any certification or any static analysis, and has been loaded into the EEPROM area,
  - This code is modified by a fault attack,
  - It becomes hostile : illegal cast to parse the memory, access to other pieces of code, unwanted call to the Java Card API (getKey,…).
- Java Virtual machine uses an offensive interpreter
  - Fault attacks are not taken into account,
  - **Java Card** Virtual Machine needs some run time checks,
  - Sometime hardware based.
- **How to characterize a good counter measure ?**
  - A complete defensive JCVM is not affordable,
  - Security level of the VM can be driven by the application;

Example of mutant

Bytecode	Octets	Java code
00 : aload_0	00 : 18	private void debit(APDU apdu) {
01 : getfield 85 60	01 : 83 85 60	
04 : invokevirtual 81 00	04 : 8B 81 00	
07 : ifeq 59	07 : 60 3B	if (pin.isValidated()) {
09 : …	09 : …	// make the debit operation
…	…	} else {
59 : goto 66	59 : 70 42	ISOException.throwIt (
61 : sipush 25345	61 : 13 63 01	
64 : invokestatic 6C 00	64 : 8D 6C 00	SW_PIN_VERIFICATION_REQUIRED);
67 : return	67 : 7A	}

Stack



Example of mutant

Bytecode

```
00 : aload_0
01 : getfield #4
04 : invokevirtual #61
07 : nop
08 : pop
09 :  …
…
59 : goto 66
61 : sipush 25345
64 : invokestatic #13
67 : return
```

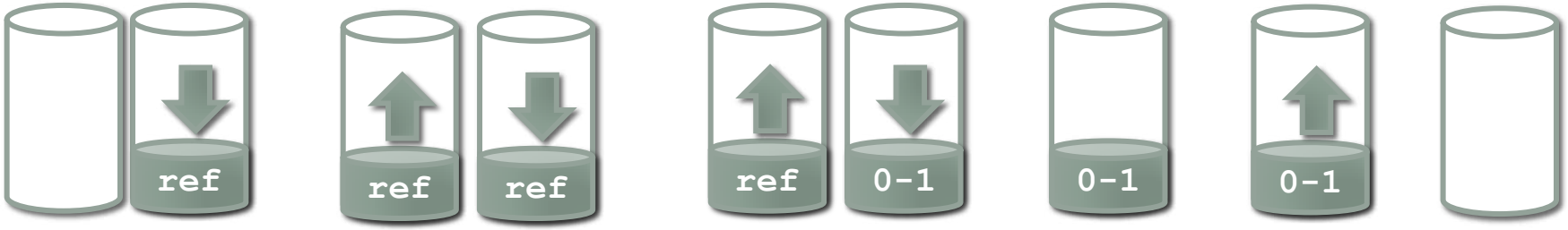
Octets

```
00 : 18
01 : 83 00 04
04 : 8B 00 3D
07 : 00
08 : 3B
09 : …
…
59 : 70 42
61 : 13 63 01
64 : 8D 00 0D
67 : 7A
```

Java code

```
private void debit(APDU apdu) {

if ( pin.isValidated() ) {

      //make the debit operation

 } else {
      ISOException.throwIt (
SW_PIN_VERIFICATION_REQUIRED);

}
}
```

Stack



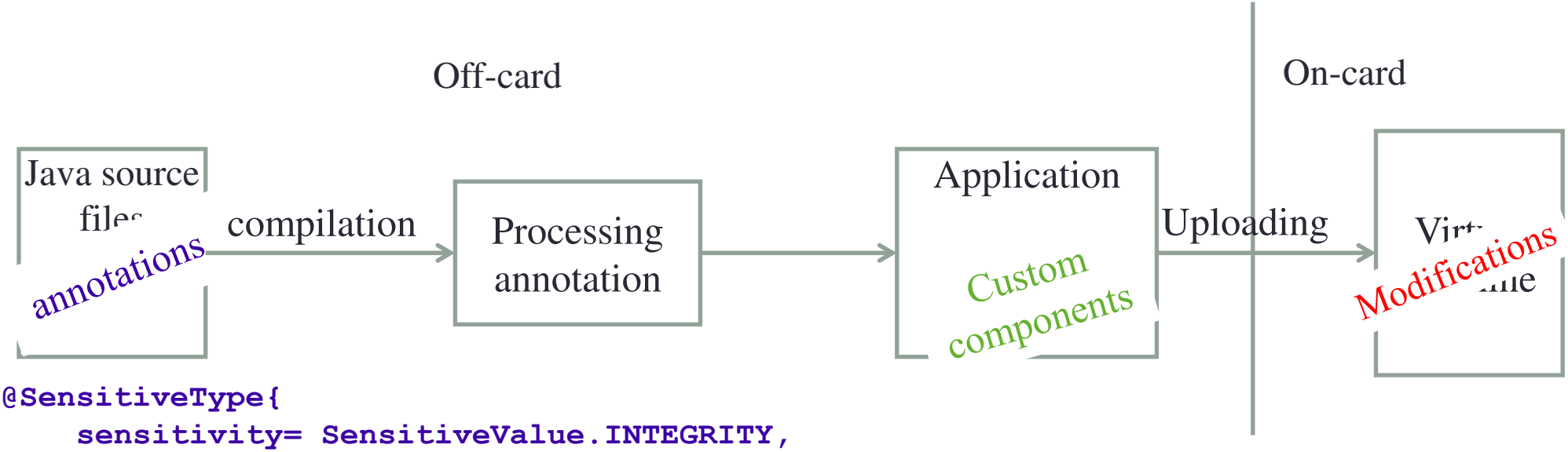
Fault models

Non-encrypted memory

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 (0x00), reset (0xFF) or random	+
Unknown byte error	loose control	byte	no control	set (0x00)or reset (0xFF) or random	-
Unknown error	no control	variable	no control	set (0x00), reset (0xFF) or random	--

Encrypted memory

Used approach




```
@SensitiveType{
    sensitivity= SensitiveValue.INTEGRITY,
    proprietaryValue="FoB"
}

private void debit(APDU apdu) {


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

```
X
X R R
X R R
X R R
...
...
X R R
X R R
X
```

Embedding CM

- Control Flow Verification
  - Detect control flow deviation
  - Principle
    - Off-card :
      - Compute all the paths using a Control Flow Graph (CFG)
      - Store the information in a custom component as a field of bits,
      - Send it with the application to the card.
    - On-card :
      - Each instruction performs a control flow check if the path is a legal one using the previously stored paths

Path Check (PCh) : example

0

```
0 aload_0;
1 getfield 4;
4 invokevirtual 18;
7 ifeq 98 (+91);
```

1

```
10 aload_1;
11 invokevirtual 11;
14 astore_2;
15 aload_2;
16 iconst_4;
17 baload;
18 istore_3;
19 aload_1;
20 invokevirtual 19;
23 i2b;
24 istore 4;
26 iload_3;
27 iconst_1;
28 if_icmpne 37 (+9);
```

2

```
31 iload 4;
33 iconst_1;
34 if_icmpeq 43 (+9);
```

3

```
37 sipush 26368;
40 invokestatic 13;
```

4

```
43 aload_2;
44 iconst_5;
45 baload;
46 istore 5;
48 iload 5;
50 bipush 127;
52 if_icmpgt 60 (+8);
```

5

```
55 iload 5;
57 ifge 66 (+9);
```

6

```
60 sipush 27267;
63 invokestatic 13;
```

7

```
66 aload_0;
67 getfield 20;
70 iload 5;
72 isub;
73 i2s;
74 ifge 83 (+9);
```

8

```
77 sipush 27269;
80 invokestatic 13;
```

9

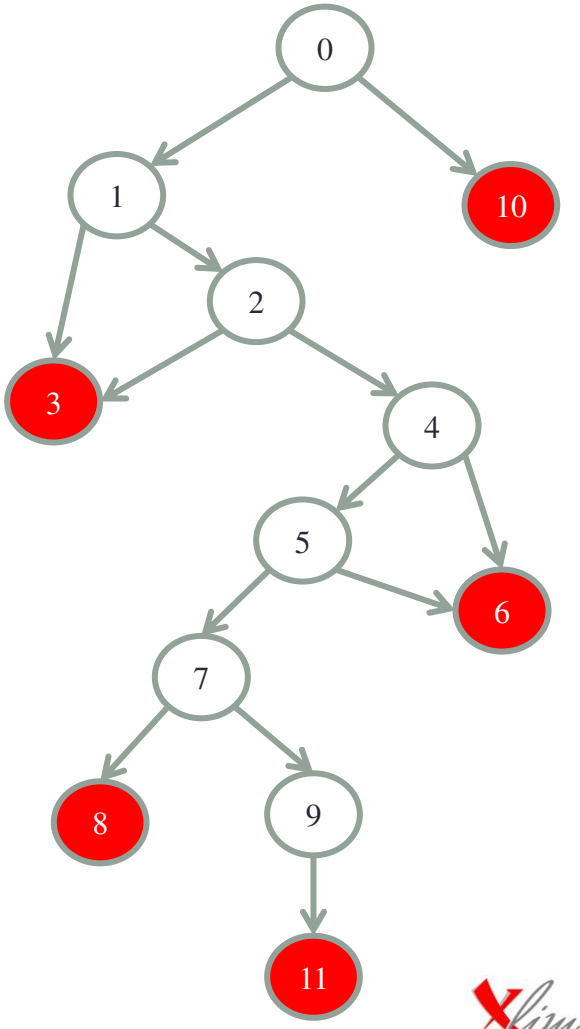
```
83 aload_0;
84 aload_0;
85 getfield 20;
88 iload 5;
90 isub;
91 i2s;
92 putfield 20;
95 goto 104 (+9);
```

10

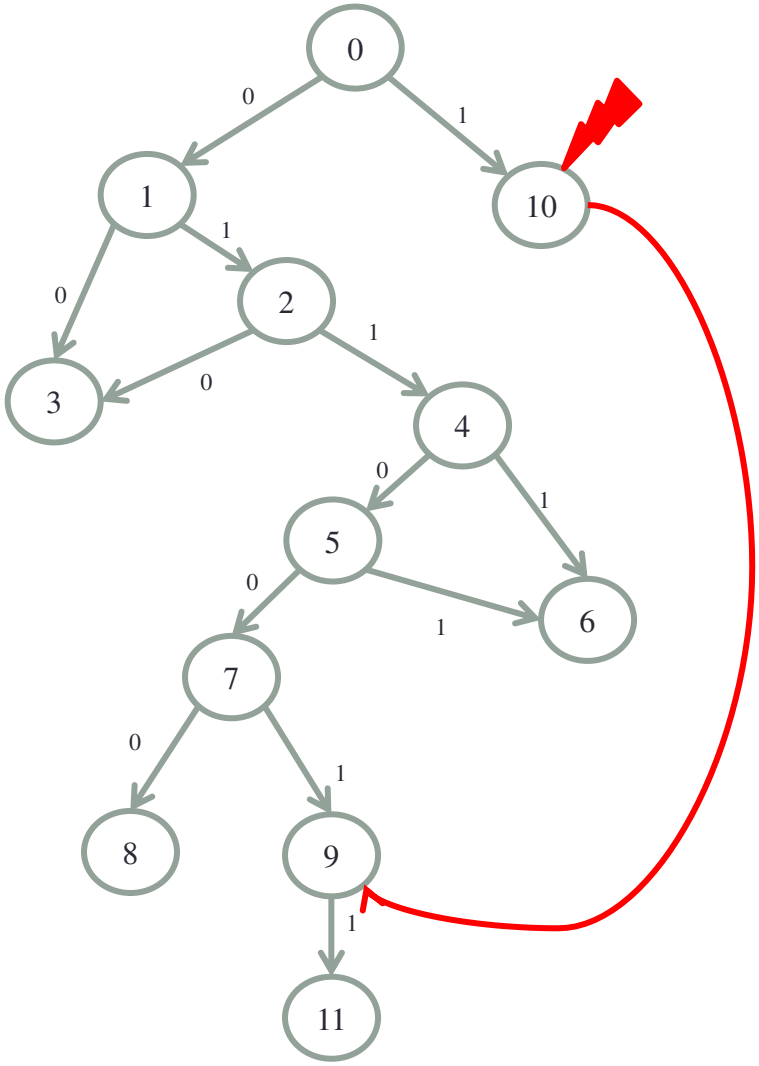
```
98 sipush 25345;
101 invokestatic 13;
```

11

```
104 return;
```



Path Check (PCh) : example



Path leading to node 9 computed off-card:

0	1	0	1	1	0	0	1

Path leading to node 9 computed on-card

0	1	1

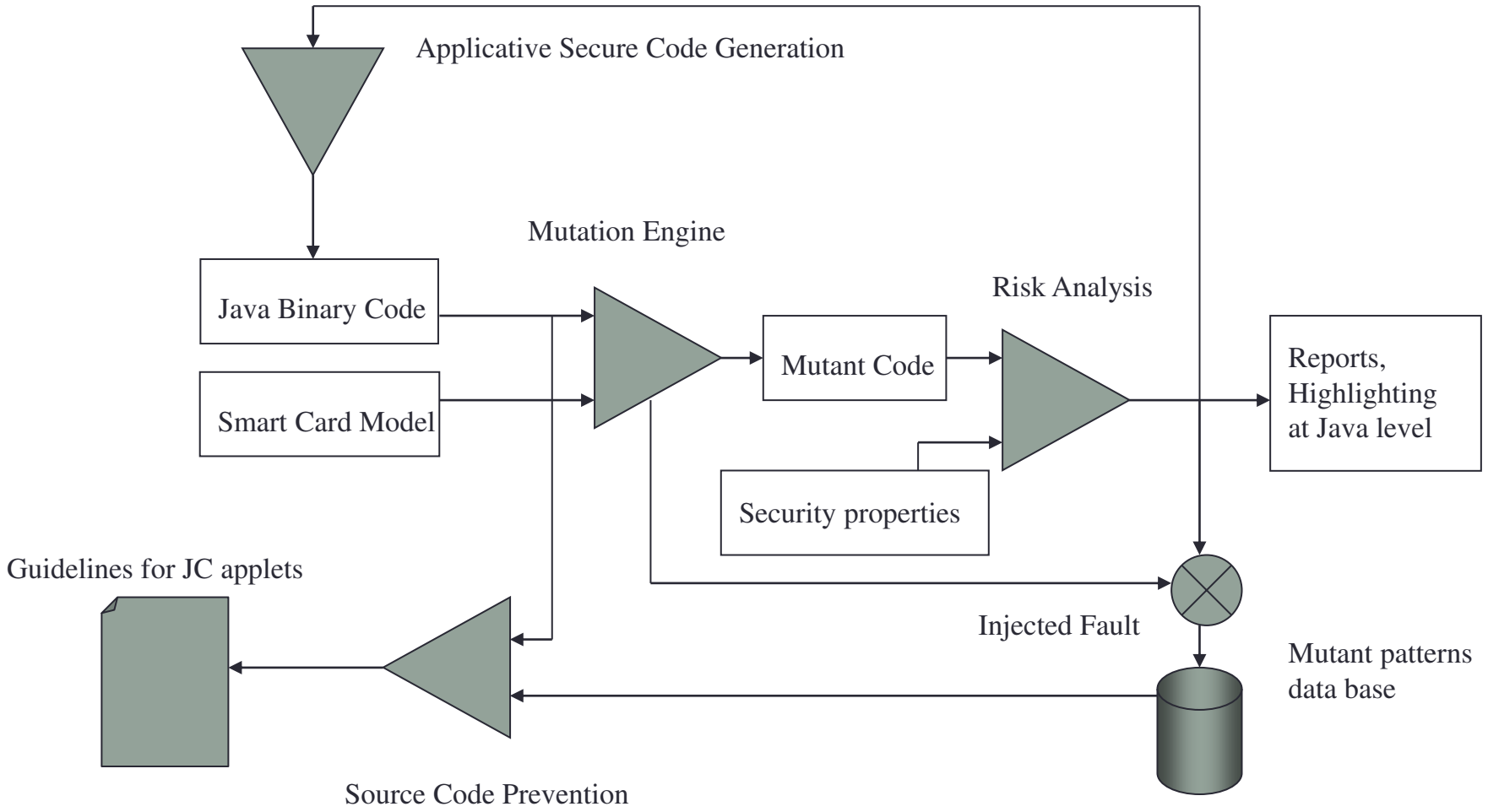
Path Check (PCh)

- Advantage
  - Allow to detect modifications that influence control flow graph and thus to fight against bypassing crucial tests.
- Drawback
  - Can’t detect a modification that doesn’t influence control flow graph .
- Evaluation of the CM
  - Efficienty,
    - Tool : fault simulator
    - Metrics : Mutant reduction, Latency, Simulation time
  - Cost memory footprint and CPU overhead
    - Modification of a JC Virtual Machine
    - Execution on a board

Design of SmartCM

- *SmartCM* investigates the ability of an application to become hostile on a given smart card platform due to a laser attack,
  - It defines several profiles corresponding to different
    - Models of smart card countermeasures,
    - Models of the attacker power,
    - Models of underlying hardware support e.g. encrypted memory,
  - It emulates the effect of the fault,
    - Only on the byte array (including the exception table) of a method not on the RTE or system variables.
    - If undetected by the CM it generates the corresponding mutant code,
    - It uses the JC 3 annotation mechanism to activate the CM,
  - It evaluates the severity of each mutant code,
    - According to a risk analysis,
  - It can automatically generates applicative CM if needed or guidelines for developers.

The Fault Simulator



Efficiency: mutants reduction

* Path Check
** Field of bit
*** Basic block

SfrOtp - 9136 attacks on 4568 instructions

Reference model
7960
-

SfrOtp	Partial BCV	PS	PCh*	FoB**	BB***
Mutant reduction	94%	95%	86%	99%	100%
Average latency	3.64	3.56	17.18	8.61	12

AgentLoc - 7008 attacks on 3504 instructions

Reference model
6486
-

AgentLoc	Partial BCV	PS	PCh*	FoB**	BB***
Mutant reduction	94%	99%	88%	99%	100%
Average latency	11.8	12.1	2.43	10.20	13

Benchmark: maximum resources consumption

* Path Check
** Field of bit
*** Basic block

	CPU overhead	EEPROM	Ram	ROM
PS	+5%	0%	≈ 0	≈ 1%
PCh*	+8%	+10%	<1%	≈ 1%
FoB**	+3%	≈ 3 %	<1%	≈ 1%
BB***	+5%	+5%	<1%	≈ 1%

Metrics obtained with all methods tagged

Conclusions

- The exposed countermeasure
  - Respectful of the Java Card specification
  - Brings security interoperability
  - Efficiency depends on the application
- It is affordable for the card
  - Memory consumption
  - CPU overhead
- Less work for developers
  - Only need to use an annotation
- Lightweight changes of the VM interpreter

Thanks you for your attention!

Any questions?

?

guillaume.bouffard@xlim.fr

http://secinfo.msi.unilim.fr