Sécurité des cartes à puce: des attaques physiques aux protections logicielles

P. Berthomé, K. Heydemann,
X. Kauffmann-Tourkestansky, **J.-F. Lalande**

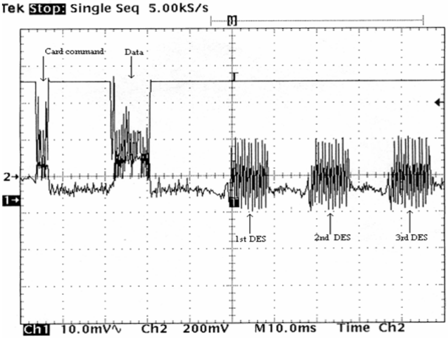
Journée Risques - 5 juin 2012

Introduction

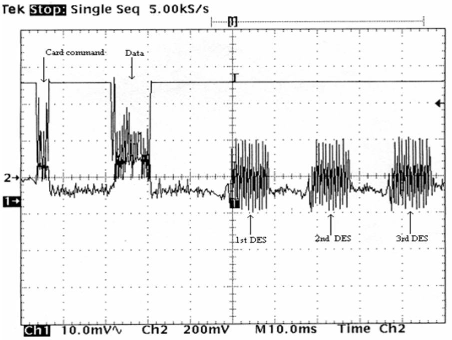
Context:

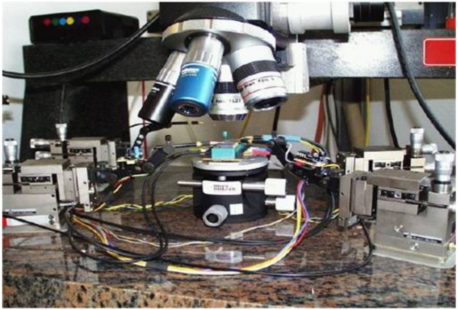
- Smart card are subject to physical attacks
- Security is of main importance for the card industry
- Adding security countermeasures
  - is not so obvious...
  - is an expensive and time consuming process

Introduction - physical attacks



Introduction - physical attacks



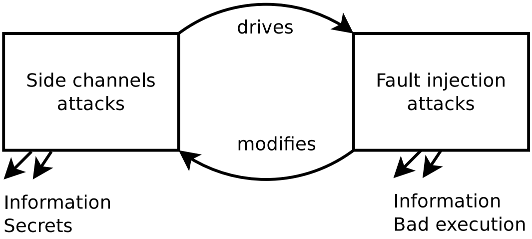


See this - Do this

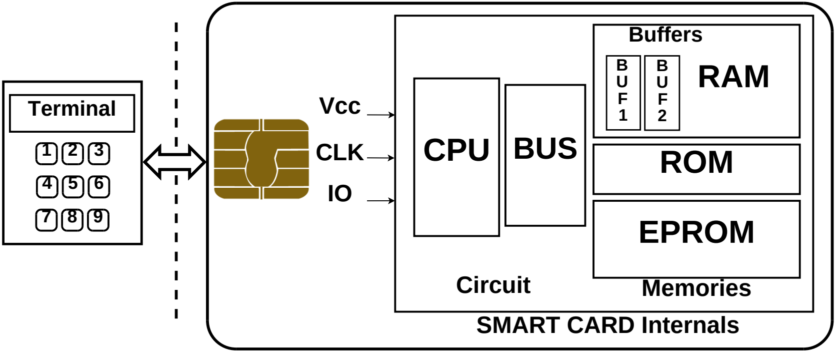
Introduction - attacks

Two types of attacks to consider:

- Side channel attacks
- Fault injection attacks



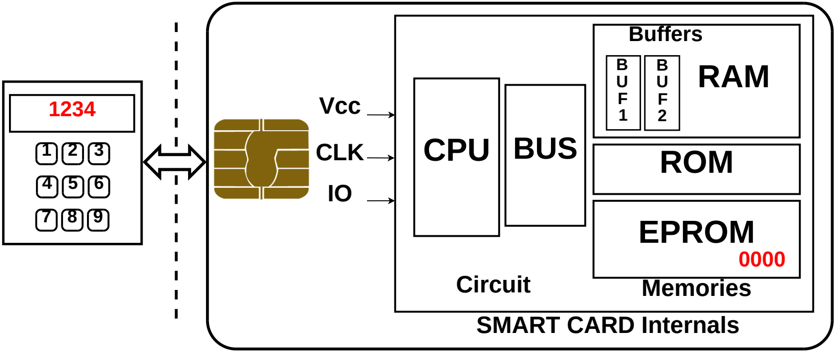
Authentication for smart card



- Information Flow -

Smart Card Internals

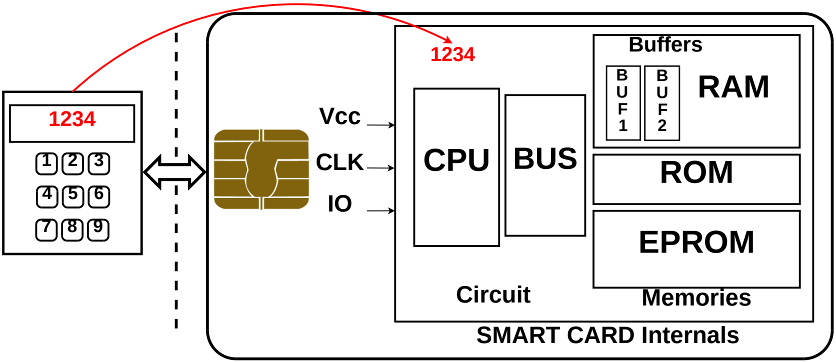
Authentication for smart card



- Information Flow -

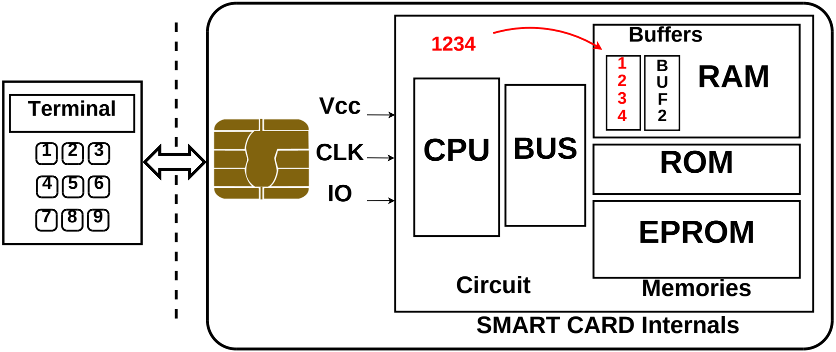
Terminal

Authentication for smart card



- Information Flow -
Sent to card's circuit

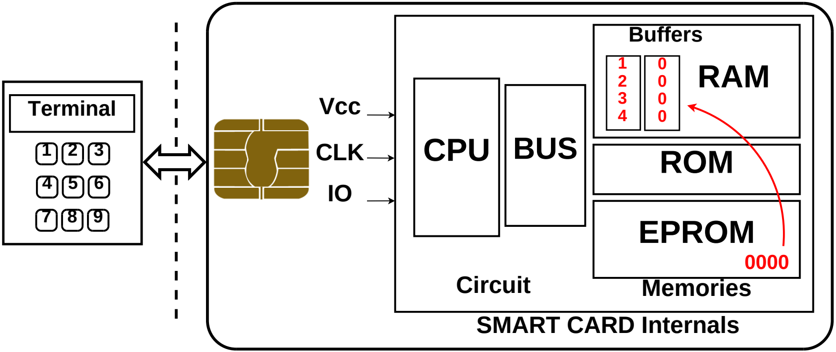
Authentication for smart card



- Information Flow -

ROM code tells CPU to load value in a RAM buffer

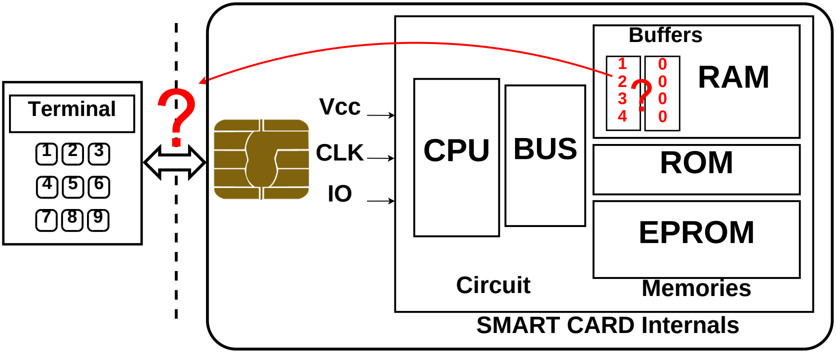
Authentication for smart card



- Information Flow -

ROM code tells CPU to load key in a RAM buffer

Authentication for smart card



- Information Flow -

ROM function compares buffers

Example of attacks

Let us consider such an authentication code:

```
uint user_tries = 0; // initialization of the number of tries for this session
uint max_tries = 3; // max number of tries
while (...) /* card life cycle: */
{
  incr_tries(user_tries);
  res = get_pin_from_terminal(); // receives 1234
  pin = read_secret_pin(); // read real pin: 0000
  if (compare(res, pin))
    { dec_tries(user_tries); }
  if (user_tries < max_tries)
    {  everything_is_fine();  }
  else
    {  killcard();  }
}
```

Simplified authentication code with pin check

Example of attacks

Let us consider such an authentication code:

```
uint user_tries = 0; // initialization of the number of tries for this session
uint max_tries = 3; // max number of tries
while (...) /* card life cycle: */
{
  incr_tries(user_tries);   → NOP ... NOP
  res = get_pin_from_terminal(); // receives 1234
  pin = read_secret_pin(); // read real pin: 0000
  if (compare(res, pin))
    { dec_tries(user_tries); }
  if (user_tries < max_tries)   // always true
    {  everything_is_fine();  }
  else
    {  killcard();  }
}
```

Simplified authentication code with pin check

Example of attacks

Let us consider such an authentication code:

```
uint user_tries = 0; // initialization of the number of tries for this session
uint max_tries = 3; // max number of tries
while (...) /* card life cycle: */
{
  incr_tries(user_tries);
  res = get_pin_from_terminal(); // receives 1234
  pin = read_secret_pin(); // read real pin: 0000
  if (compare(res, pin))
    { dec_tries(user_tries); }
  if (user_tries < max_tries)
    {  everything_is_fine();  }
  else
    {  killcard();  }   → NOP ... NOP
}
```

Simplified authentication code with pin check

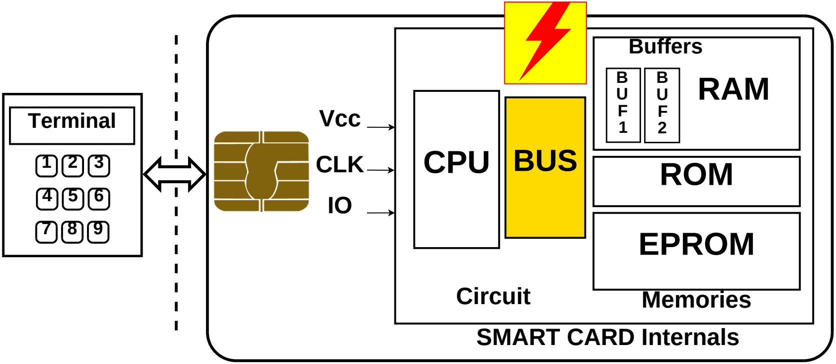
Example of attacks

Let us consider such an authentication code:

```
uint user_tries = 0; // initialization of the number of tries for this session
uint max_tries = 3; // max number of tries
while (...) /* card life cycle: */
{
  incr_tries(user_tries);
  res = get_pin_from_terminal(); // receives 1234
  pin = read_secret_pin(); // read real pin: 0000   pin ← "1234"
  if (compare(res, pin))   // always true
    { dec_tries(user_tries); }
  if (user_tries < max_tries)
    {  everything_is_fine();  }
  else
    {  killcard();  }
}
```

Simplified authentication code with pin check

Attack vectors



Example: attack on card bus!

What about security?

Security problems

Several questions appear:

- How to explain low level attacks at source code level?
- How to identifiy harmfull attacks?
- How to implement countermeasures?
- How to evalute the efficiency of countermeasures?

Security problems

Several questions appear:

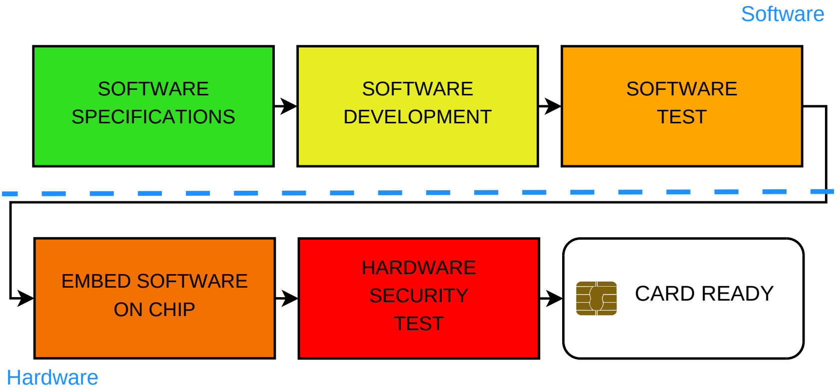
- How to explain low level attacks at source code level?
- How to identifiy harmfull attacks?
- How to implement countermeasures?
- How to evalute the efficiency of countermeasures?

Two goals

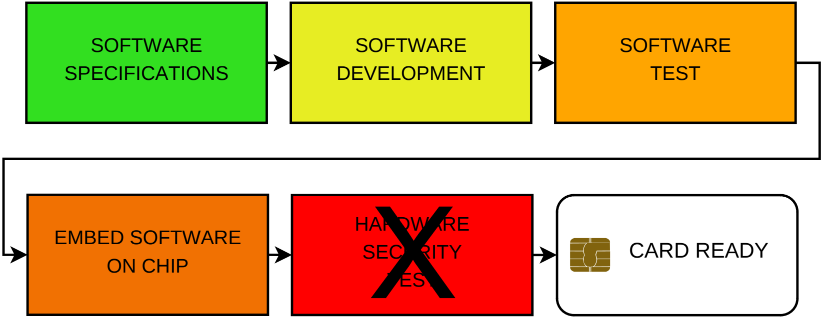
- Create a high level model of attacks (developer level)
- Provide a security test methodology

Outline

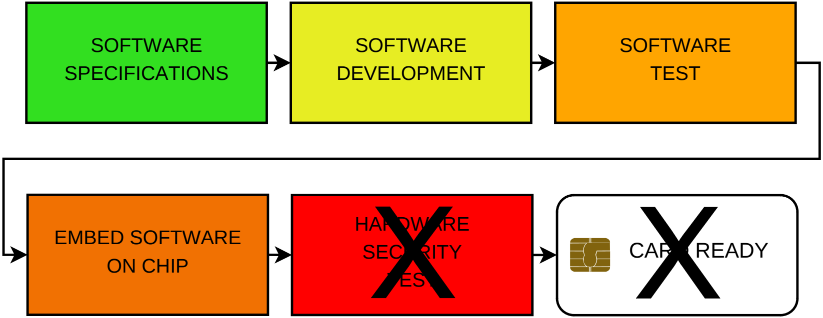
Smart Card Development Process



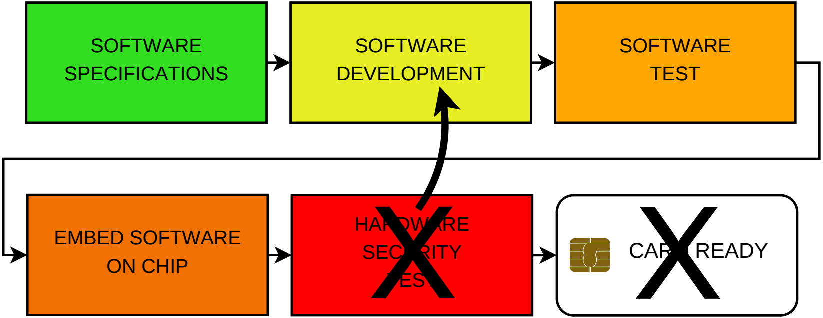
Smart Card Development Process



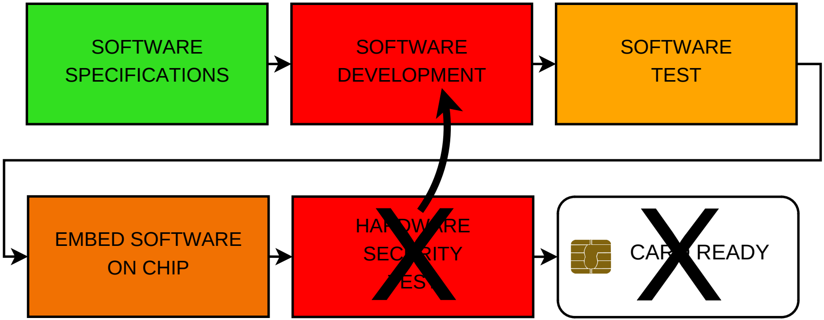
Smart Card Development Process



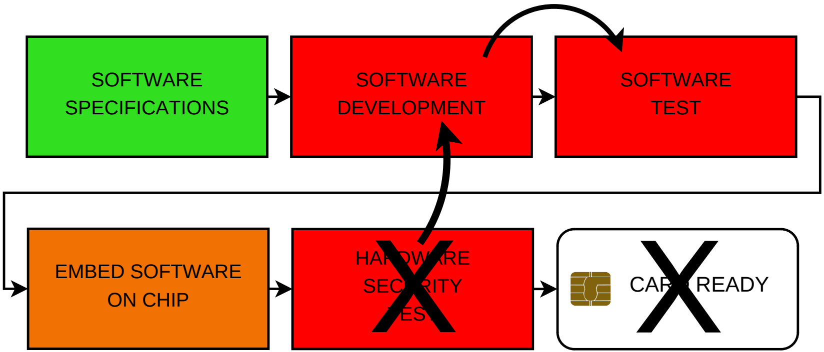
Smart Card Development Process



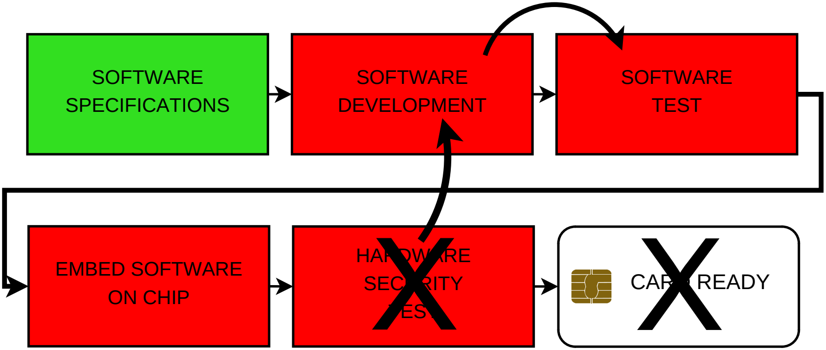
Smart Card Development Process



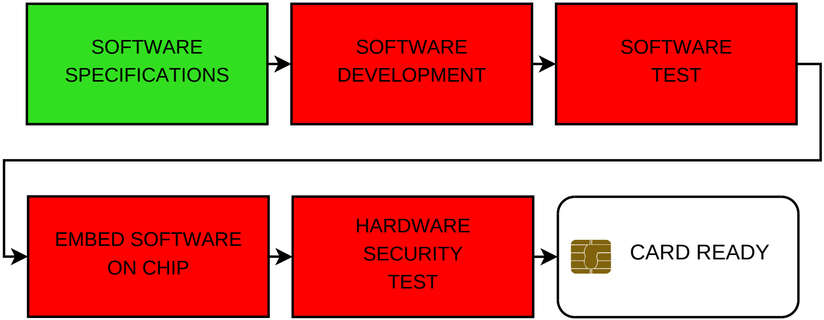
Smart Card Development Process



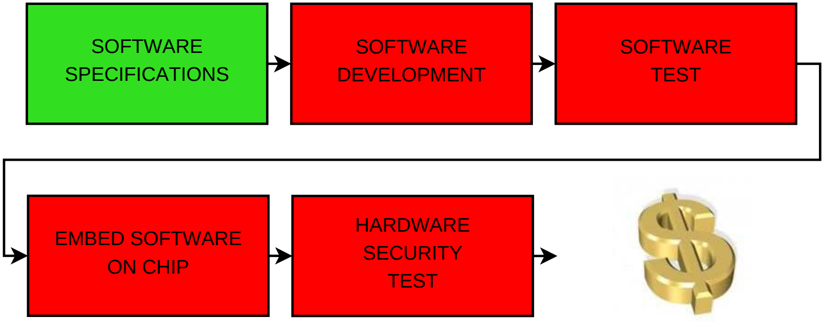
Smart Card Development Process



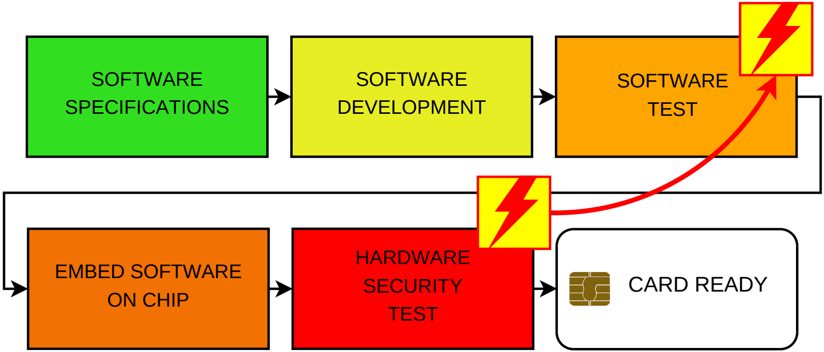
Smart Card Development Process



Smart Card Development Process



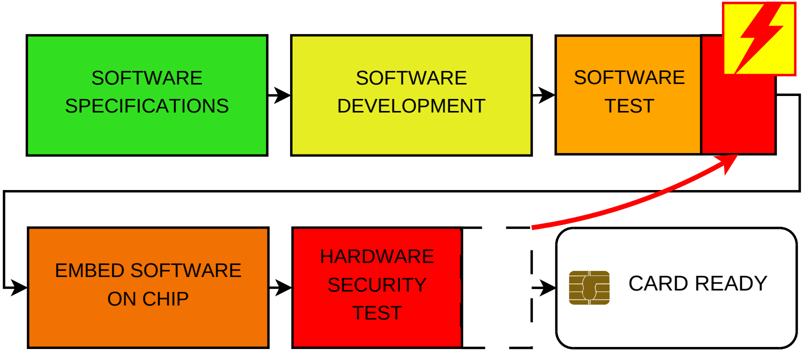
Smart Card Development Process



Objectives:

- Simulate hardware attacks at software level
- Move some security hardware tests to software level

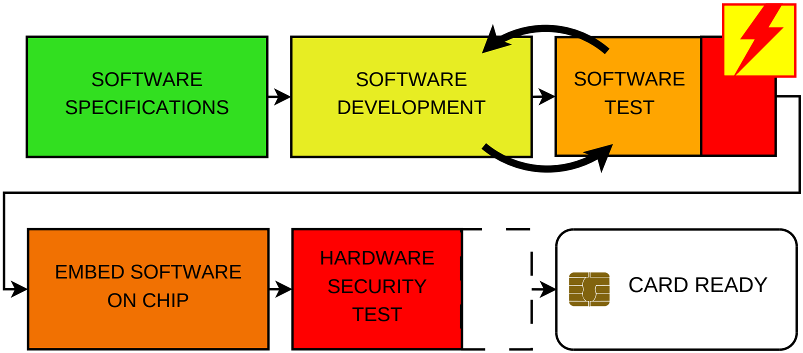
Smart Card Development Process



Objectives:

- Simulate hardware attacks at software level
- Move some security hardware tests to software level

Smart Card Development Process



Objectives:

- Simulate hardware attacks at software level
- Move some security hardware tests to software level

Attack hypothesis

Attacks on smart cards in Common Criteria [1]:

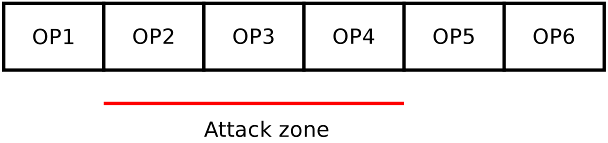
- modifying a value read from memory;
- changing the quality of random numbers generated;
- **modifying the program flow**.

Attack model in the literature [2]:

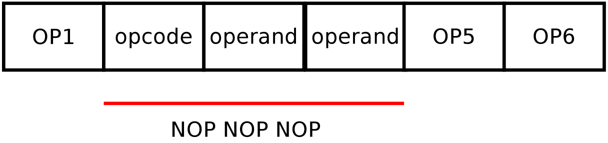
- precise bit error;
- **precise byte error**;
- unknown byte error;
- unknown error.

Attack hypothesis

One or several consecutive bytes are overwritten:



- bytes encode operations that are opcodes or operands
- for example, one opcode and its operands may be deleted:



Hypothesis and difficulties

Hypothesis:

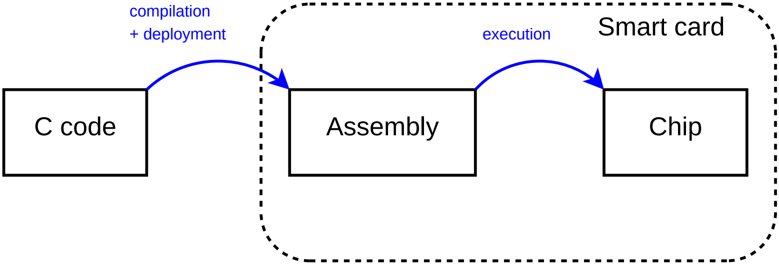
- One attack during one execution
- One attack on one or several consecutive bytes

Difficulties:

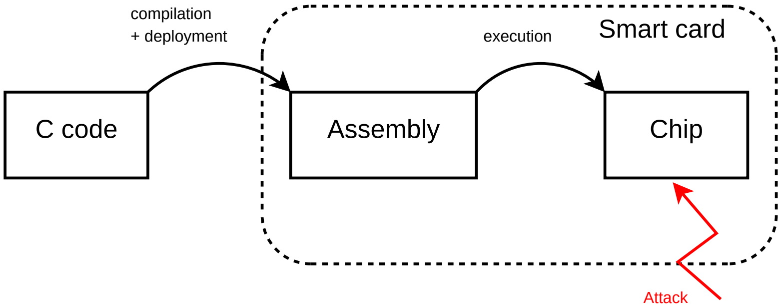
- What happens when an opcode is deleted?
- What happens when an operand is deleted?
- What happens when an opcode is replaced?

Outline

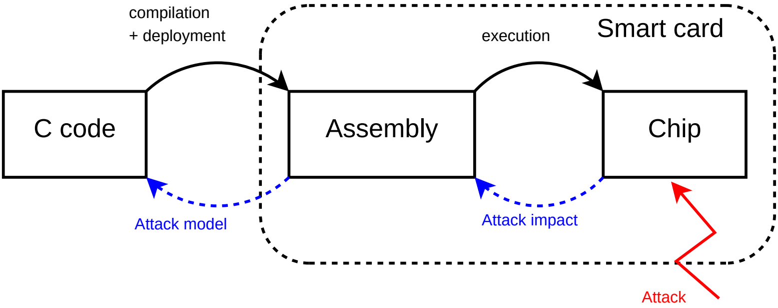
Example of high level model of a low level consequence



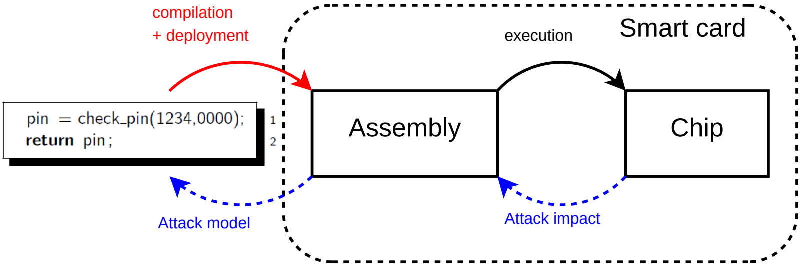
Example of high level model of a low level consequence



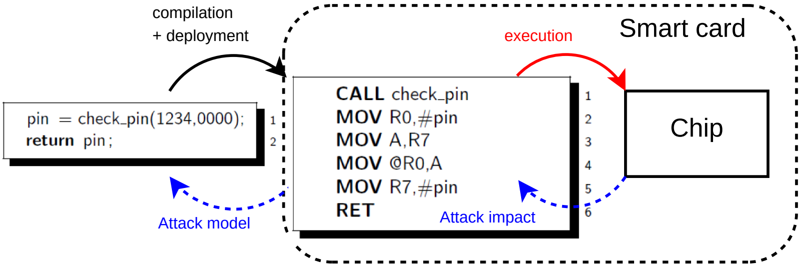
Example of high level model of a low level consequence



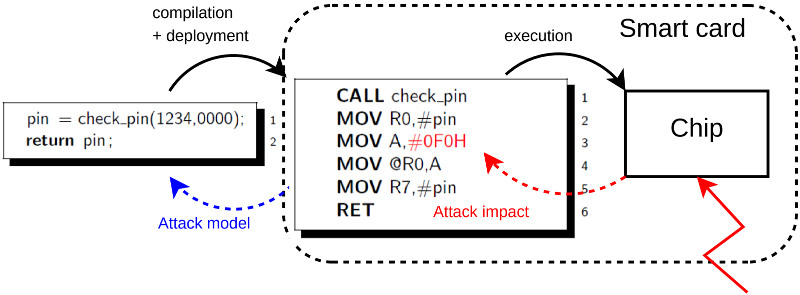
Example of high level model of a low level consequence



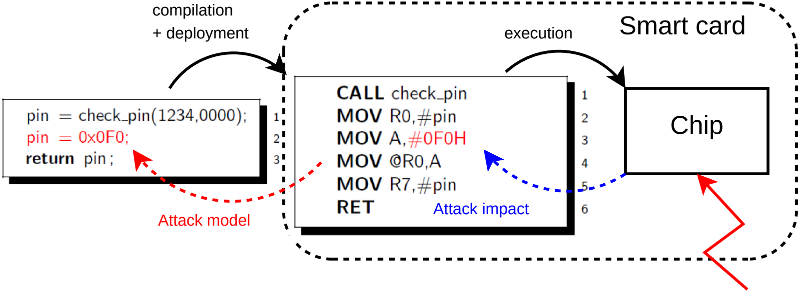
Example of high level model of a low level consequence



Example of high level model of a low level consequence



Example of high level model of a low level consequence



Flow shifting for opcode replacement

Opcode X becomes opcode Y

X arg1 arg2 ; → Y arg1 ; arg2

- arg2 is viewed as an opcode and the instruction flow is shifted

Flow shifting for opcode replacement

Opcode X becomes opcode Y

X arg1 arg2 ; → Y arg1 ; arg2

- arg2 is viewed as an opcode and the instruction flow is shifted

X arg1 arg2 ; → Y ; arg1 arg2

- arg1 is viewed as an opcode. Depending on the number of bytes needed by arg1, arg2 is then either an operand or an opcode. The instruction flow has also shifted.

Flow shifting for opcode replacement

Opcode X becomes opcode Y

X arg1 arg2 ; → Y arg1 ; arg2

- arg2 is viewed as an opcode and the instruction flow is shifted

X arg1 arg2 ; → Y ; arg1 arg2

- arg1 is viewed as an opcode. Depending on the number of bytes needed by arg1, arg2 is then either an operand or an opcode. The instruction flow has also shifted.

X arg1 arg2 ; → Y arg1 arg2 ;

- the instruction flow has not shifted.

But... shifted flows quickly recover

Lemma: A shifted flow recovers to the normal flow in $1/p$ operations, with p the probability that a random byte is an opcode in the original flow.

- Using the 8051 assembly code, $p = 0.64$.
- The flow recovers in 1.56 steps...

Are attacks always successful ?

- new opcodes may crash the program
- the original opcodes may suffer from missing opcodes

Assembly attack consequences

Let us take the following example:

```
c = u + 5;
b = c < 10;
if (b){
 res = c + 1;
}
else{
 res = 0;
}
```

```
    mov r2,dpl      // load the parameter in r2
    mov a,#0x05     // put 5 into a
    add a,r2        // compute u + 5 in a
    mov _c,a        // store c into RAM from a
    clr c           // clear the carry
    subb a,#0x0A    // computes b i.e. c-10
    jnc 00102$      // jumps to 102
                    // if carry is not set
    mov a,_c        // load c into a
    inc a           // a++ i.e c + 1
    mov r2,a        // r2 stores a (res = c + 1)
    sjmp 00103$     // jump over else
00102$:
    mov r2,#0x00    // r2 stores 0 (res = 0)
00103$:
    mov dpl,r2      // push r2 on the stack
```

Example 1: NOP insertion

```
    mov r2,dpl
    mov a,#0x05
    add a,r2
    mov _c,a
    clr c
    subb a,#0x0A → NOP
    jnc 00102$
    mov a,_c
    inc a
    mov r2,a
    sjmp 00103$
00102$:
    mov r2,#0x00
00103$:
    mov dpl,r2
```

```
c = u + 5;
b = c < 10;
if (b){ → if(false)

  res = c + 1;
}
else{
  res = 0;
}
```

Example 1: NOP insertion

```
    mov r2,dpl
    mov a,#0x05
    add a,r2
    mov _c,a
    clr c
    subb a,#0x0A → NOP
    jnc 00102$
    mov a,_c
    inc a
    mov r2,a
    sjmp 00103$
00102$:
    mov r2,#0x00
00103$:
    mov dpl,r2
```

```
c = u + 5;
b = c < 10; goto label;
if (b){

  res = c + 1;
}
else{ label :
  res = 0;
}
```

Example 2: NOP insertion (again !)

```
    mov r2,dpl
    mov a,#0x05 → NOP
    add a,r2
    mov _c,a
    clr c
    subb a,#0x0A
    jnc 00102$
    mov a,_c
    inc a
    mov r2,a
    sjmp 00103$
00102$:
    mov r2,#0x00
00103$:
    mov dpl,r2
```

```
c = u + 5;→ c = u+?
b = c < 10;
if (b){

  res = c + 1;
}
else{
  res = 0;
}
```

Example 2: NOP insertion (again !)

```
    mov r2,dpl
    mov a,#0x05 → NOP
    add a,r2
    mov _c,a
    clr c
    subb a,#0x0A
    jnc 00102$
    mov a,_c
    inc a
    mov r2,a
    sjmp 00103$
00102$:
    mov r2,#0x00
00103$:
    mov dpl,r2
```

```
c = attack();
b = c < 10;
if (b){

  res = c + 1;
}
else{
  res = 0;
}
```

Example 3: instruction override

```
    mov r2,dpl
    mov a,#0x05
    add a,r2
    mov _c,a
    clr c
    subb a,#0x0A
    jnc 00102$
    mov a,_c
    inc a
    mov r2,a
    sjmp 00103$
00102$:
    mov r2,#0x00
00103$:
    mov dpl,r2 → jmp 102
```

```
c = u + 5;
b = c < 10;
if (b){
 res = c + 1;
}
else{
 label:
 res = 0;
}
goto label;
```

High level attack model

The examples show that

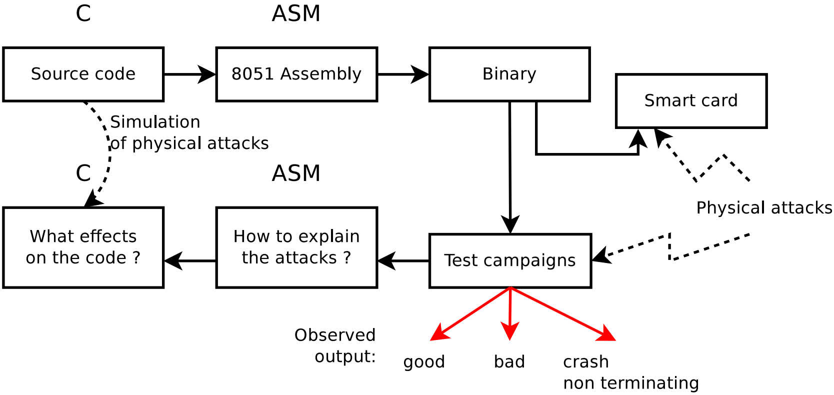
- the variables may be affected
- the flow control may be changed
- arbitrary jumps may be introduced

The high level attack model proposed is based on:

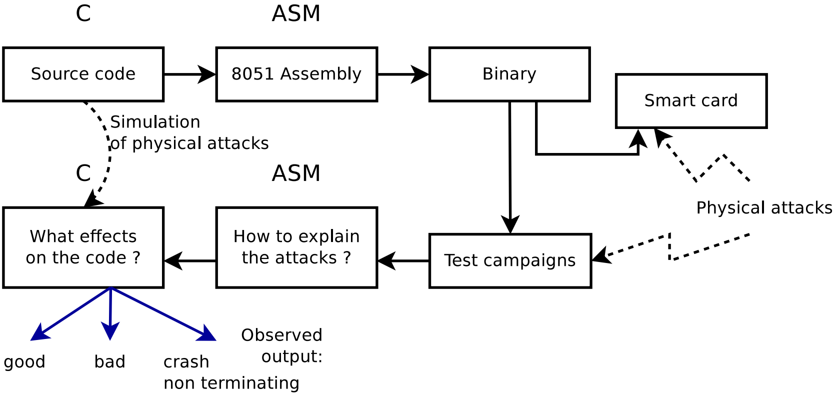
- perturbating variables: $a = attack();$
- introducing inconditional jumps: *goto label;*

Outline

Principles of experiments



Principles of experiments



Principles of experiments

Principles:

- Generate high-level attacks: new C source codes
- Test exhaustively the resulting programs

How to classify attack effects ?

- Good: the execution gives the expected output
- Bad: the ouput is wrong or an error occured
- Crash: the program crashed
- Signal: a signal has been received (SIGSEGV)
- Killed: an infinite loop occured

Good candidates

Candidates:

- Pure C programs with measurable input/output
- ⇒ SPEC 2006 benchmark suite
- Jump attacks stay into a function
- Bzip2:
  - 107 functions, 8 643 C statements
  - assembly code: 26 103 instructions

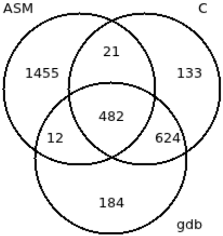
Bzip2	Assembly code	High level C
Source code size (lines)	26 103	8 643
Nb generated attacked codes	3 531 954	117 802

n^2 attacks for each bzip2 function of size n

Coverage

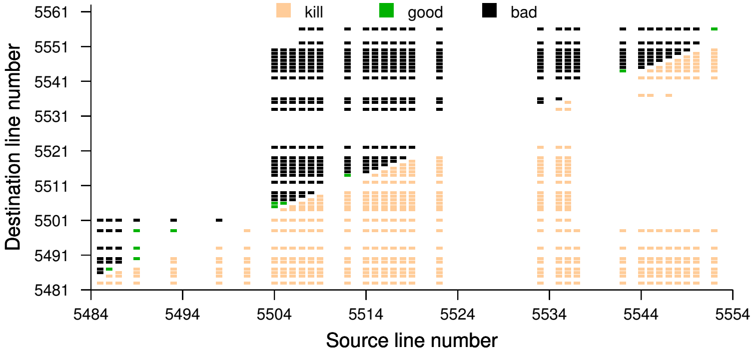
Statistics	ASM	C	gdb
Code size	26 103	8 643	8 643
Simu. time	2d 18h	8h	2h
Nb of BADs	273 129	14 050	5 417
Uniq BADs	2 326	1245	852
ASM coverage	100%	21%	21%

Statistics for simulated attacks on bzip2



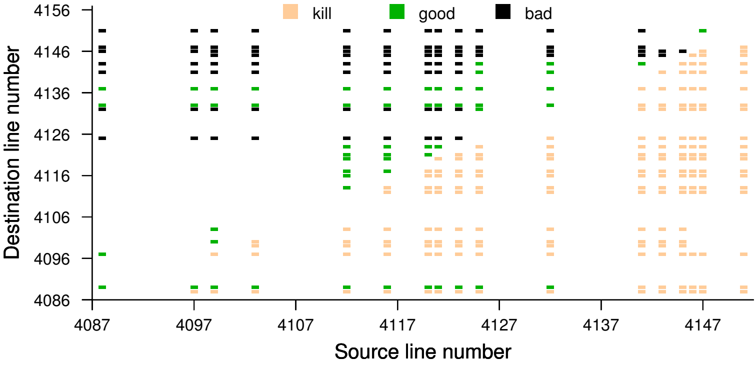
Coverage of uniq BADs

BZ2_compressBlock profile



Spatial classification of good/bad/kill attacks according to source/dest. lines, simulated in C against BZ2_compressBlock

BZ2_blockSort profile

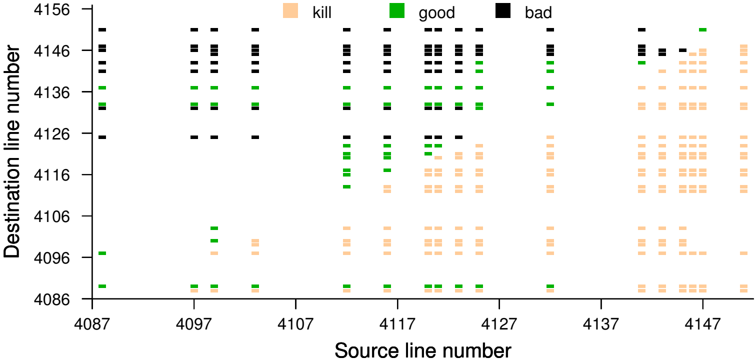


Spatial classification of good/bad/kill attacks according to source/destination lines, simulated in C against BZ2_blockSort

Implementing countermeasures

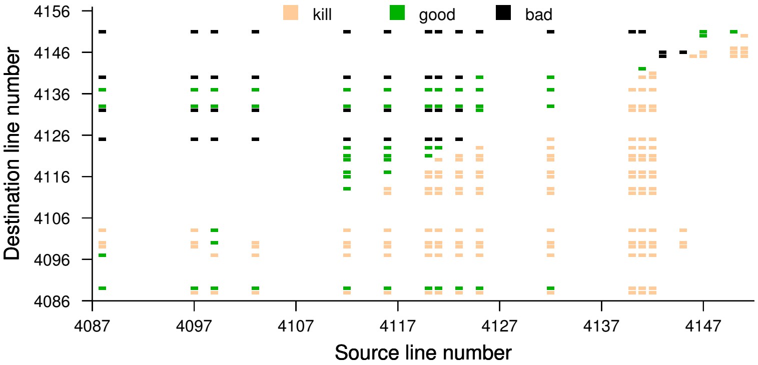
```
Original code                                      /** Countermeasure **/
                      /** kc(); = {perror("KILLCARD");exit(-1);} **/
                                                   int security = 48;
...
                                                          security++;
s->origPtr = -1;
                              if (security != 49) kc(); security++;
for (i = 0; i < s->nblock; i++)
{                           if (security != 50+2*i) kc(); security++;
  if (ptr[i] == 0) {
    s->origPtr = i;
    break;
  };                        if (security != 51+2*i) kc(); security++;
}
                                              if (security < 50) kc();
AssertH(s->origPtr != -1, 1003);
```

Implementing countermeasures: before



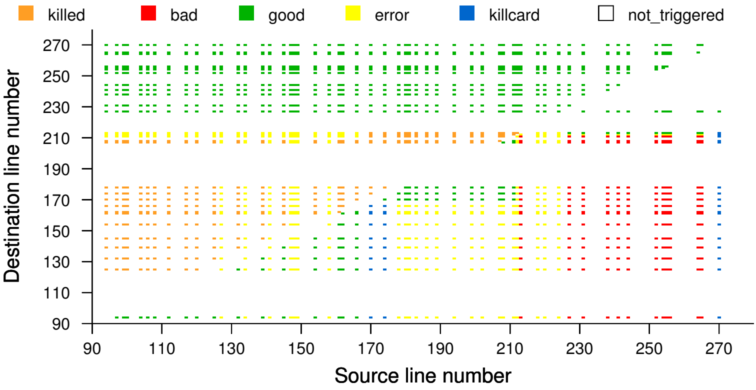
BZ2_blockSort: before

Implementing countermeasures: after



BZ2_blockSort: after

Results for an sensitive function of a smart card



Spatial classification for a sensitive function of a smart card code

Conclusion

Problems

- How to model physical attacks at software level?
- How to inject jump attacks?
- How to classify the impact of attacks?

Contributions

- Attack injection platform for C programs
- Experimental results on bzip2 and smart card codes
  - Profiling of attacks
  - Identification of weak points in functions

Questions

Common Criteria.
Application of Attack Potential to Smartcards.
Technical Report March, BSI, 2009.

A. A. Sere, J. Iguchi-Cartigny, and J.-L. Lanet.
Automatic detection of fault attack and countermeasures.
In 4th Workshop on Embedded Systems Security, pages 1–7, New York city, New York state, USA, 2009. ACM Press.