ANR-ASTRID SERTIF : Simulation for the Evaluation of Robustness of embedded Applications against Fault injection

ANR-14-ASTR-0003-01

http://sertif-projet.forge.imag.fr/

Marie-Laure Potet[1], Jessy Cledière[2], Thanh-Ha Le[3]

(1) Laboratoire VERIMAG, Université de Grenoble-Alpes
(2) CEA-LETI
(3) SAFRAN IDENTITY AND SECURITY

11 octobre 2016

Context

⇒ Secure components (Hardware and Software) providing security services (authentification, cryptography) and secure storage of information.

- Attractive targets for attackers
- Can be physically attacked

⇒ Must be protected against high level attack potential (AVA-VAN.5)

Agence Nationale de la Recherche ANR

DGA

ASTRID

Fault injection

- Perturbation attacks (EM or laser) $\Longrightarrow$ fault injection.
- Fault injection modifies the control and data flows.

```c
int verify(char buffer[4]) {
    int i;
    int authenticated = 1;
    // comparison loop
    for(i = 0; i < 4; i++) {
        if(buffer[i] != pin[i]) {
            authenticated = 0;
        }
    }
    // CM: redundant check
    if (i != 4) { // CM
        muteCard();
    }
    return authenticated;
}
```

```
MOV R0, #00h ; i = 0
MOV R3, #01h ; authenticated = 1
JMP WHILE
DO:
MOV R2, [buffer+i]
MOV A, [pin+i]
CMP A, R2
JE  ITER ; buffer[i] == pin[i]?
MOV R3, #00h ; authenticated = 0
ITER:
INC R0 ; i++
WHILE:
MOV A, R0
CMP A, #04h
JB DO ; i < 4?
MOV A, R0
CMP A, #04h
JNE muteCard ; i != 4?
MOV A, R3
RET
```

Fault injection

- Perturbation attacks (EM or laser) $\Longrightarrow$ fault injection.
- Fault injection modifies the control and data flows.

```c
int verify(char buffer[4]) {
    int i;
    int authenticated = 1;
    goto ATTACK;
    for(i = 0; i < 4; i++) {
        if(buffer[i] != pin[i]) {
            authenticated = 0;
        }
    }
    ATTACK:
    if (i != 4) { // CM
        muteCard();
    }
    return authenticated;
}
```

```asm
MOV R0, #00h ; i = 0
MOV R3, #01h ; authenticated = 1
JMP WHILE
DO:
MOV R2, [buffer+i]
MOV A, [pin+i]
CMP A, R2
JE  ITER ; buffer[i] == pin[i]?
MOV R3, #00h ; authenticated = 0
ITER:
INC R0 ; i++
WHILE:
MOV A, R0
CMP A, #04h
NOP
MOV A, R0
CMP A, #04h
JNE muteCard ; i != 4?
MOV A, R3
RET
```

Fault injection

- Perturbation attacks (EM or laser) $\Longrightarrow$ fault injection.
- Fault injection modifies the control and data flows.

```
int verify(char buffer[4]) {
    int i;
    int authenticated = 1;
    // comparison loop
    for(i = 4; i < 4; i++) {
        if(buffer[i] != pin[i]) {
            authenticated = 0;
        }
    }
    // CM: redundant check
    if (i != 4) { // CM
        muteCard();
    }
    return authenticated;
}
```

```
MOV R0, #04h ; i = 0
MOV R3, #01h ; authenticated = 1
JMP WHILE
DO:
MOV R2, [buffer+i]
MOV A, [pin+i]
CMP A, R2
JE ITER ; buffer[i] == pin[i]?
MOV R3, #00h ; authenticated = 0
ITER:
INC R0 ; i++
WHILE:
MOV A, R0
CMP A, #04h
JB DO ; i < 4?
MOV A, R0
CMP A, #04h
JNE muteCard ; i != 4?
MOV A, R3
RET
```

Assessing Robustness Against Fault Injection

Is an embedded application robust against fault injection?

- **Penetration Testing:** Physical perturbation attacks on the application under test to **inject faults**.
  - Look for successful attacks (=compromising security).
  - Factors for Attack Potential Calculation
- **Code Analysis:** Detect vulnerabilities in the application with a **code review**.
  - Look for attack paths using a given fault model.
  - Originally manual process, now with automatic tools
  - Success rate $\mathcal{T} = \frac{\mathcal{F}_S}{\mathcal{F}}$.

Factor
Elapsed Time
Expertise
Knowledge of the TOE
Access to the TOE
Equipment
Open Samples

Table: Factors of Attack Potential

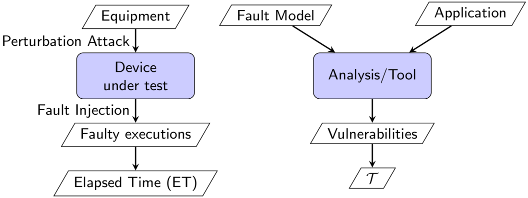


Figure: The 2 processes

Sertif objectives

Consortium:

- CEA-LETI: J. Clédière, L. Dureuil, Ph. de Choudens, C. Dumas
- SAFRAN Identity and Security: Thanh-Ha Le, Ch. Cachelou, A. Crohen, L. Rivière
- Vérimag: ML Potet, L. Mounier, G. Petiot

Main objective: rationalize and automate as much as possible the robustness assessment process (for evaluator and developer) w.r.t. the state-of-the-art (spatial and temporal multiple faults) including reproductivity and re-evaluation.

More concretely:

- Combination between physical attacks and code review
- Simulation tools evaluation (including robustness criteria)
- Evaluation of countermeasure relevance

Open problems ... and some results

- A better articulation between code review and penetration testing
  - How to link code vunerabilities with penetration test and vice versa?
  - how to be confident in the used fault model?

⇒ Cardis 15, Lionel Rivière PhD thesis, Louis Dureuil PhD thesis (next talk)

⇒ ...

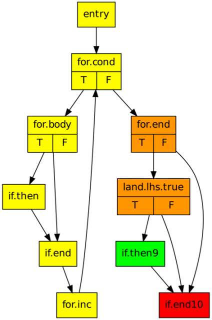
- Code analysis by tools
  - Automatisation: a reproductible, complete and timeless process
  - Generally a combinatorial process producing a lot of attacks
  - Measures of robustness?

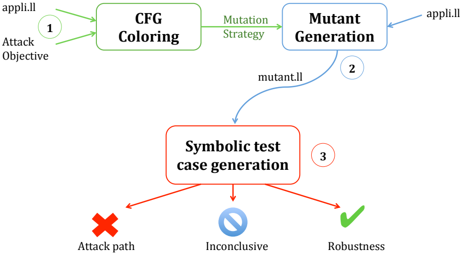
⇒ 3 types of tools: Lazart (Vérimag), CELTIC (CEA), EFS (SAFRAN) and the FISSC benchmark

⇒ ...

Lazart (Vérimag)

⇒ C code robustness evaluation against fault injection using symbolic execution





- Fault model: condition inversion, skip call, data modification
- Goal: Reach or avoid a CFG block or a logical formula
- Possibility of multiple fault injection scenarios

Lazart (2)

⇒ a high-level tool dedicated to logical weakness in the algorithms.

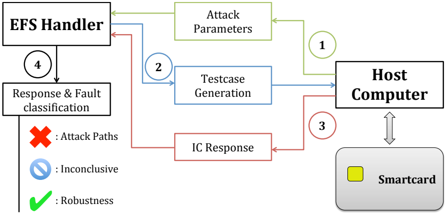
- An interactive tool (to play with fault injection): symbolic inputs, oracles and fault models
- Based on Klee, a concolic tool for LLVM. Potentially activates all possible paths and fault injections.
- A notion of redundant attacks (fault injection points)
- Scenario representation in terms of graphs

Verifypin_2 example:

#*fault injection*	#*attacks*	#*non redundant attacks*
1	2	2
2	9	1
3	19	0
4	21	1

EFS (SAFRAN Identity and Security)

- **E**mbedded **F**ault **S**imulator: An embedded tool within the target device (e.g. smartcard), running at Hardware Abstraction Layer.



- Fault mechanism: a subroutine with a high priority level, granting read/write access to all the component registers and memories.
- Fault models: allows arbitrary code to be executed in an interruption (e.g. register value modification, RAM modification, instruction skipping/replacement, arbitrary jumps. . .).
- Advantages:
  - fault injections on physical component.
  - side-channel observations.

EFS (2)

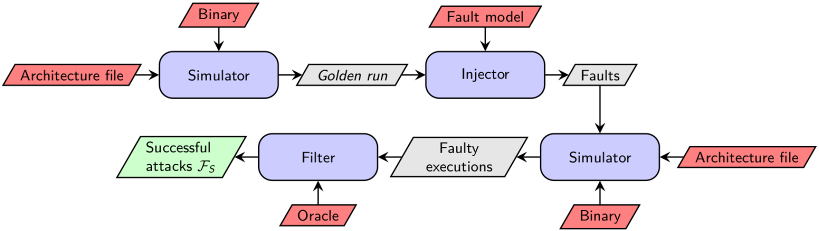
Results obtained with the EFS:

- For each of the execution cycle of the targeted routine(s), we collect:
  - The routine(s) response
  - The address of the attacked instruction
- An externalized Oracle analyses the responses
- Results on AES last round with fault model $PC \leftarrow PC + 2$

	Fault rate	
Fault type	without CM	with CM
No attack	4.683 %	4.683 %
Board reboot	5.785 %	6.336 %
Coutermeasure activated	0.0 %	88.430 %
One byte difference on output	76.309 %	0.0 %
2 to 15 bytes differencies on output	0.275 %	0.0 %
Random output	9.091 %	0.551 %

CELTIC (by CEA-LETI)

Native smartcard binaries simulation with fault injection.

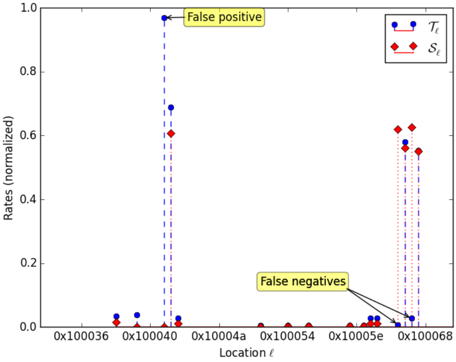


- Custom Architecture Description Language for retargetability.
- Exhaustive injection campaign at the binary level
- Fault models: base library extensible with scripts (fault model composition)
- User-defined victory oracles.
- JIT-enabled simulation for improved performance

CELTIC (2)

CELTIC Outputs:

- Execution trace for the *Golden Run*
- The list $\mathcal{F}_S$ of successful attacks.
- For each successful attack:
  - Characteristics of the fault (address, instant, type of fault)
  - Faulty execution trace



FISSC: our secure collection

⇒ a Fault Injection and Simulation Secure Collection

Objectives:

- Evaluation of simulation tools
- Evaluation of (hardened) implementations

Difficulties:

- No available collected examples
- Tools dedicated to various fault models and levels of code
- How to compare results? Attacks?

Our proposal:

- A collection of (extensible) examples
- High level attack scenarios with regard to success oracles
- Matching criteria between results (by address or by fault model)

Contents

Examples:

Example	Oracle
VerifyPIN	`g_authenticated == 1`
VerifyPIN	`g_ptc >= 3`
AES KeyCopy	`! equal(key, keyCpy)`
GetChallenge	`equal(challenge, prevChallenge)`
CRT-RSA	`(g_cp == pow(m,dp) % p && g_cq != pow(m,dq) % q)`
	`|| (g_cp != pow(m,dp) % p && g_cq == pow(m,dq) % q)`

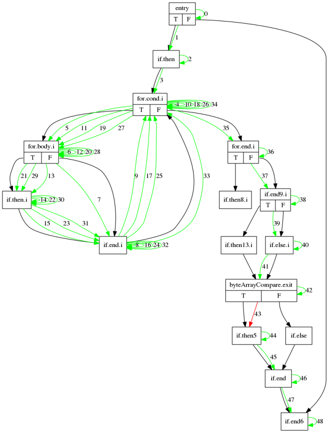
Countermeasures: hardened booleans, virtual stack, double arguments, step counter, loop counter, data redundancy, double calls, double tests, control flow integrity

Programming Features: Explicit call, Fixed Time Loops, inlining

Results

- Normalized and modular examples
- C sources and Thumb-2 assembly listings
- high-level attack scenarios on CFG

Example	1-fault atk	2-fault atk
VerifyPIN	2	0
+fixed time loops	2	1
+FTL +inlining	2	1
+FTL +INL +loop counter	2	0
+FTL +double calls	0	4
+FTL +INL +double tests	0	3
+FTL +INL +DT +step counter	0	2
+control flow integrity	0	2
+FTL +INL +DT +SC +CFI	0	1



CFG for 'verifyPIN_2' function

Using the benchmark

- **Get** `http://sertif-projet.forge.imag.fr/`
- **Analyze** C sources, asm listings
- **Compare** your results against the archived results
- **Contribute** your examples, countermeasures and results

⇒ An example with results using CELTIC and EFS:
`http://sertif-projet.forge.imag.fr/pages/example.html`

A first piece…

HL scenario coverage

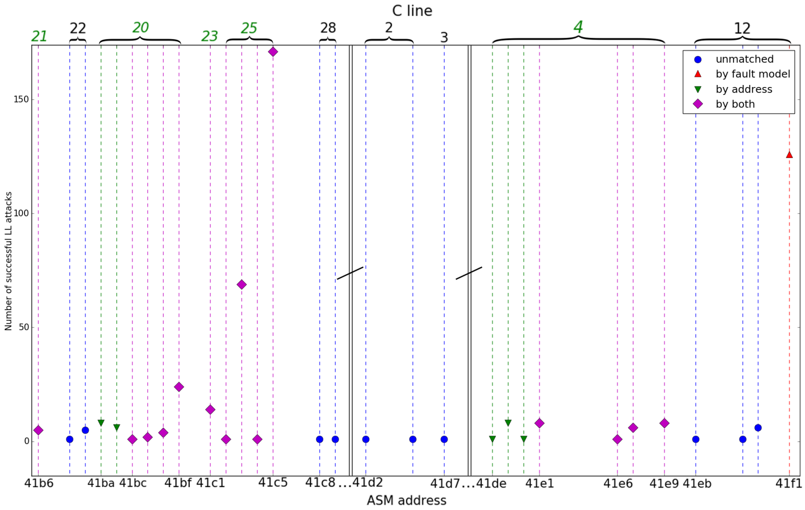


Figure: Matching HL and LL attacks

An open problem: Fault Injection Code Metrics

⇒ How results can be evaluated?

- Identify sensitive points in a code
- Propose a vulnerability rate (evaluator's point of view). For instance:

$$\frac{|successful\ attack|}{|realized\ attacks|}$$

- Determine how to harden the code (developer's point of view): regroup "equivalent" attacks

Metrics difficulties:

- Attacker's model
- sensibility to the size of code

An open problem: Countermeasures analysis

Objectives:

- How to choose adapted countermeasures ?
  - depend on the fault model
  - could be costly
  - complexity due to multiple fault injection (CM can be attacked)

Open problems:

- Define and test metrics against various hardened examples
- Cost and comparison between classical countermeasures
- Dedicated analysis to establish dependency between contermeasures and assets to be protected
- ...

An open problem: a process mixing code analysis and penetration testing

With a good knowledge of possible attacker's parameter for a given device is it possible to mainly use simulation tools?

- How to determine precisely an attacker model for a given device?
  - component characterization against EM, laser, FBBI...
  - how to reveal only flash modification, registers modifications from RAM modifications, during data transfer or its storage ...
- A more reproductible and automatic process compatible with a certification process?

References

Louis Dureuil: Analyse de code et processus d'évaluation des composants sécurisés contre l'injection de fautes. PhD thesis, October 2016

L. Dureuil, G. Petiot, M-L. Potet, T-H. Le, A. Crohen and P. de Choudens. FISSC: a Fault Injection and Simulation Secure Collection SAFECOMP 2016.

L. Dureuil, M-L. Potet, P. de Choudens, C. Dumas and J. Clédière. From Code Review to Fault Injection Attacks: Filling the Gap using Fault Model Inference. Cardis 2015.

Lionel Rivière. Securing software implementations against fault injection attacks on embedded systems. PhD thesis, TELECOM ParisTech, Paris, September 2015.

L. Rivière, M-L. Potet, T-H. Le, J. Bringer, H. Chabanne and M. Puys. Combining High-Level and Low-Level Approaches to Evaluate Software Implementations Robustness Against Multiple Fault Injection Attacks. FPS 2014

L. Rivière, Z. Najm, P. Rauzy, J-L. Danger, J. Bringer, Laurent Sauvage: High precision fault injections on the instruction cache of ARMv7-M architectures. HOST 2015

M-L. Potet, L. Mounier, M. Puys and L. Dureuil. Lazart: a symbolic approach to evaluate the impact of fault injections by test inverting. ICST 2014, International Conference on Software Testing.

M. Berthier, J. Bringer, H. Chabanne, T-H. Le, L. Rivière, V. Servant. Idea: Embedded Fault Injection Simulator on Smartcard. ESSoS 2014