Tools and Benchmark for robustness code evaluation against fault injection

Marie-Laure Potet and Lionel Morel

VERIMAG, Grenoble, France
CEA-Dacle, Grenoble, France

21 mai 2018

Challenges :

⇒ How to build and evaluate applications robust against fault injection attacks ?

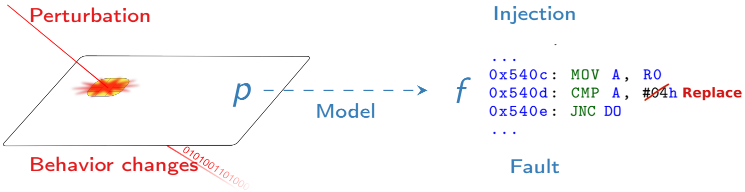
- Reproductible evaluation processes :
  - tools adaptable to new fault models and attack technics
  - evaluation process adaptable to the considered context (smartcard, secure element, Iot, TEE, . . .) and expected level of assurance

- Spatial and temporal multi-fauts as a the state-of-the-art requiring to to revisit :
  - fault model combination and representative attacks
  - helping developpers to chose adapted counter-measures
  - result analysis and robustness evaluation metrics

Our works

- A whole process for helping vulnerability analysis (CEA Cesti/VERIMAG)
- FISCC : a Fault Injection and Simulation Secure Collection (project ANR-DGA ASTRID 2014)
- Lazart : a public tool based on symbolic execution for helping developers and auditors
- Adding ccounter-measures at the compiling time (CEA-Dacle)
- A new type of application and domain : attacking secure boots (project IRT Nanoelec CLAPS)

From Perturbation Attack to Fault Injection



- Attacker cannot choose the fault in code with precision

$$f \triangleq (i = 124, \text{store}([\texttt{0x540d}], 0))$$

- Only chooses the parameters of the equipment

$$p \triangleq (x = 12\,\mu\text{m},\ y = 24\,\mu\text{m},\ d = 3800\,\text{ns}, w = 850\,\text{ns})$$

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 $\mathcal{T} = \frac{\mathcal{F}_S}{\mathcal{F}}$.

Elapsed time	Rating
$<$ one hour	0
$<$ one day	3
$<$ one week	4
$<$ one month	6
$>$ one month	8
Not practical	_

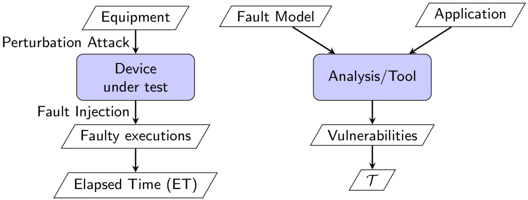
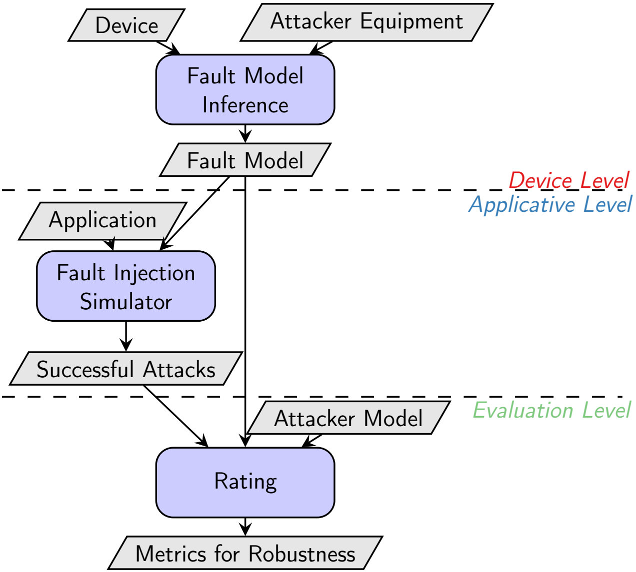


Figure – The 2 processes

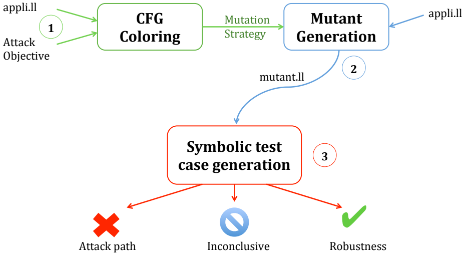
The Louis Dureuil's thesis end-to-end Approach



Lazart (1)

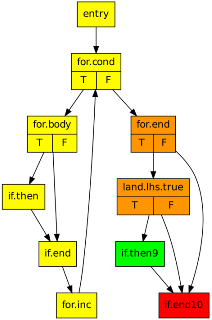
⇒ C code robustness evaluation against fault injection based on symbolic execution

- a single mutant embbeding fault models and fault injections
- guided by a goal : reach or avoid a CFG block or a logical formula
- supporting multiple faults and several (potentially symbolic) fault models
- strategies to inject faults depending on the fault model and goals.



Lazart (2)

- A notion of redundant attacks (fault injection points)
- Scenario representation in terms of graphs
- Could be used for countermeasures analysis



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

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)

Exemple	Reach CM (1F)	Attaques (1F)	Reach return ($\neg CM$ et $\neg Auth$)		Nb appels CM
			0F	1F	
$VPIN_0$	N/A	2	1	0	0
$VPIN_1$	1	2	1	2	1
$VPIN_2$	5	2	1	5	1
$VPIN_3$	5	2	1	5	1
$VPIN_4$	8	2	1	5	5
$VPIN_5$	7	0	1	5	2
$VPIN_6$	7	0	1	5	3
$VPIN_7$	17	0	1	5	13

$\Rightarrow$ Could be extended to the point where countermeasures are raised.

FISSC : an open source secure collection

Content :

- A collection of (extensible) examples
- High level attack scenarios with regard to success oracles

<table>
<tr><th>Example</th><th>Oracle</th></tr>
<tr><td>VerifyPIN</td><td><code>g_authenticated == 1</code></td></tr>
<tr><td>VerifyPIN</td><td><code>g_ptc >= 3</code></td></tr>
<tr><td>KeyCopy</td><td><code>! equal(key, keyCpy)</code></td></tr>
<tr><td>GetChallenge</td><td><code>equal(challenge, prevChallenge)</code></td></tr>
<tr><td>CRT-RSA</td><td><code>(g_cp == pow(m,dp) % p && g_cq != pow(m,dq) % q)</code>
<code>|| (g_cp != pow(m,dp) % p && g_cq == pow(m,dq) % q)</code></td></tr>
</table>

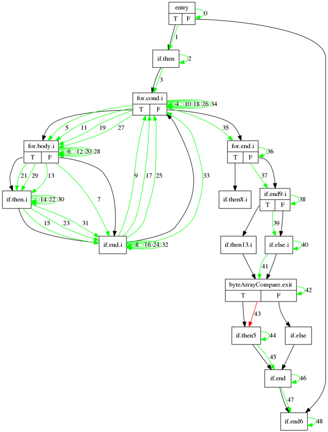
Countermeasures : hardened booleans, virtual stack, double arguments, step counter, loop counter, data redundancy, double calls, double tests, control flow integrity

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	2-fault
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