Motivation — Fault injection (FI) [1] is a common threat for embedded systems. Security analysis under FI attacks can be performed either using tests on real platforms [2], [3] or using simulation [4], [5] and formal methods [6], [7] considering an ISA-level fault model such as instruction skipping or register corruption. However, it has been shown that many fault effects are not directly addressable at the ISA level [8] and that precise knowledge of the microarchitecture is essential for a better evaluation. We propose a solution to model both software (SW) and hardware (HW) parts of a system and then to formally evaluate its robustness to FI attacks. This approach is complementary to the analysis methods used at the software level. It identifies non-visible faults at the ISA level that affect the security of the software and that leverage the specificities of a microarchitecture.

Workflow to Study the Effects of FI — Fig. 1 illustrates the workflow we propose to analyse the effects of FIs. It requires as input: 1) the hardware description of the evaluated microarchitecture, i.e., both its combinatorial and sequential logics (Verilog 2005 and SystemVerilog 2017 are supported), 2) the executable program, i.e., the instructions to be executed and their associated data, 3) the fault model which describes the localization (both in time and space) and the effects of the attack (e.g., bit flip, bit set) on the hardware, and 4) the security property.

The formal model is a transition system produced by the RTL synthesis tool Yosys and described with the SMT-LIB language. It is constrained by assumptions which describe the execution of the program and the effect of the FIs while the assertions express security properties. Vulnerabilities are found with the Yices SMT Solver using bounded model checking techniques. Returned counter-examples highlight the propagation of the faults in the microarchitecture.

The developed workflow can formally check the execution of a hundred instructions in the presence of faults and is not suitable for larger programs. It is also possible to extract a sequence of instructions from a program and check its robustness with degrees of freedom in the initial state of the system.

TABLE 1: faults injected found by our analysis over VerifyPin.

<table>
<thead>
<tr>
<th>Module</th>
<th>Wire</th>
<th>Timing</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>rd_controler</td>
<td>operand_a_fw_mux_sel_o</td>
<td>@57-</td>
<td>bit-flip</td>
</tr>
<tr>
<td>prefetch_buffer</td>
<td>status_cnt_n</td>
<td>@21-26</td>
<td>bit-set</td>
</tr>
</tbody>
</table>

References


