10 Software-Activated Hardware Trojans

Exploiting a Vulnerable CPU with Software

A Introduction

With a remarkable progression in electronic designs, integrated circuits (ICs) are involved in our day-to-day lives. We use consumer electronics like printers, scanners, personal computers, projectors, etc., in everyday use, making IC security extremely critical. Most of the ICs are fabricated by off-shore foundries due to financial purposes. In addition, the companies receive services like design and testing, intellectual property (IP) supply from third parties. So, this provides an adversary with several opportunities to thwart the security aspects of the IC. **Hardware Trojan (HT)** is such a major security concern to the general security and trust of any electronic system, which is defined as the intentional malicious modifications to an electronic device that can alter device functionality or result in undesired performance. An HT-infected integrated circuit (IC) may experience functional instability, leak secret information, or provide inconsistent performance – whatever the Trojan designer intended. Hardware Trojans are designed by attackers to be stealthy and evade detection by normal testing means, and can be inserted at any phase of the IC development. These factors make finding and eliminating hardware Trojans a complex problem.

B Background

Trojans which impact circuit functionality generally have two fundamental components: the trigger and payload. A Trojan trigger monitors different internal signals and combines these signals in some way (e.g. through combinational logic) or tracks how many times the combination of signals has been seen (e.g. with sequential logic / counters). Once the condition is met, the *payload* is deployed, which can be some activity that modifies the circuit behavior. The payload is activated only when the trigger detects the expected event or sequence of events that meet some condition. Hence, from the attacker's perspective, it is desirable for the trigger condition to occur under extremely rare events, so the payload will be inactive most of the time, and the IC appears to be Trojan-free. For Trojans whose triggering conditions is a simple combination of internal signals, the trojan can be classified as *combinational*, whereas those that require a specific series of conditions to be met are classified as *sequential*. An example of a combinational Trojan may be the ANDing of multiple rare nodes, e.g. $T = (X_0 \wedge X_1 \wedge X_2)$, where X_0, X_1, and X_2 rarely transition from 0 to 1, and the condition that all three are simultaneously 1 is even more rare. In a sequential Trojan, the T may drive the clock input of a counter, so that this event must occur n times before the payload is deployed.

Trojans can be designed to modify the circuit functionality; as an **example,** consider a device in Figure 1(a) with a crypto module inside it, which produces ciphertext after encrypting the input with the stored secret keys. As long as the keys are kept secret, the ciphertext doesn't contain any input information. An attacker may want to get the secret keys out of the device, and it can be done with small modifications to the design. Figure 1(b) shows an example of a combinational Trojan-inserted design with an additional MUX and a trigger circuit. The MUX outputs either the ciphertext (select=0) or the keys (select=1) based on the trigger circuit output. The trigger circuit here produces either a logic high or a logic low based on a specific input to the device. The particular input is only known to the attacker, and considering a 128-bit of input line, the probability of that

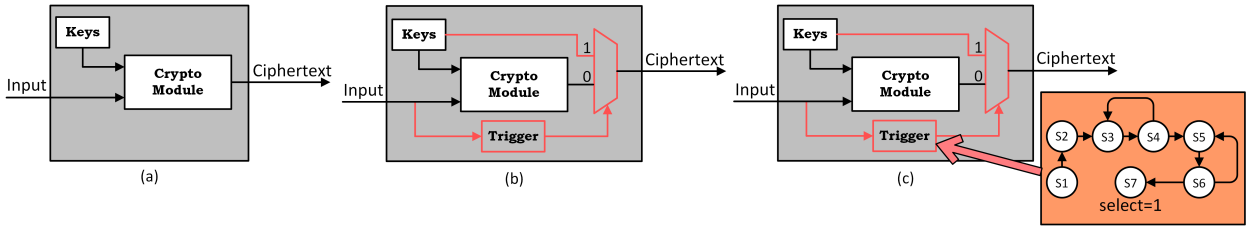


Figure 1: (a) A Trojan-free design. (b) An example of a combinational Trojan in the design. (c) An example of a sequential Trojan in the design.

specific input is $1/2^{128}$, which is extremely low. So, this Trojan will be exceptionally challenging to detect during the testing process as it is tiny and will remain inactive most of the time. Another example of a sequential Trojan-inserted design is shown in Figure 1(c). Here, the trigger circuit will generate a logic high (select=1) based on a specific sequence of events. For example, the state machine's different states can be different inputs, and when this particular sequence of inputs is applied, the trigger circuit generates the logic high signal.

The Central Processing Unit (CPU) is responsible for carrying out a computer program's instructions by executing arithmetic and logic operations. In general, the fundamental components of a CPU are: (1) Control Unit (2) Arithmetic and Logic Unit (ALU) (3) Registers. The control unit of the CPU comprises the controlling circuits, which directs the entire system to execute the program instructions. The control unit doesn't execute the instructions itself, but it satisfies all the CPU requirements to execute the instructions. The arithmetic and logic unit (ALU) contains the electronic circuits that execute all the arithmetic and logical operations. Another essential part of the CPU is the registers which are the temporary storage areas for data or instructions. The registers are not a part of the main memory or RAM but are the additional storages that offer a faster CPU execution.

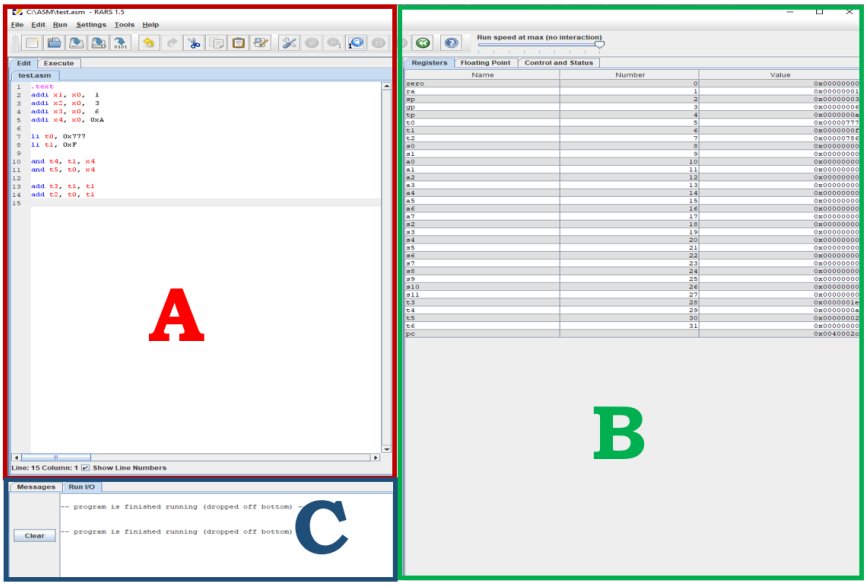


Figure 2: RISC-V simulator: (A) Editor window for entering assembly programs. (B) All the contents of RISC-V registers. (C) Output window.

C Hardware and Software Tools

The experiment's required tools are RISC-V Assembler and Run-time Simulator (RARS) and ISim, Xilinx ISE's built-in simulator.

RISC-V Assembler and Runtime Simulator: RARS (RISC-V Assembler and Runtime Simulator) is a Java-based simulator for the RISC-V instruction set architecture (ISA). Ensure you have the latest Java runtime environment (JRE) installed. You should be able to launch RARS using the *jar* file to launch the simulator GUI. This simulator is used to assemble and simulate RISC-V assembly language programs' execution. The GUI has two tabs; the **Editor** tab is to enter the assembly program, and the **Execute** tab simulates the assembly program entered in the editor. To simulate, you must first assemble your code (Figure 2(a)). The simulator allows you to run the current program, run one step at a time, undo the last step, reset memory/registers, and dump the machine code to binary strings. The right window (Figure 2(b)) shows the contents of the CPU registers. The output window (Figure 2(c)) will not be used because printing to the console requires environment calls (ecalls) that are not supported by the target CPU. Try running the sample assembly code provided (*test.asm*) by opening the file in the editor, assembling, and running step by step. See the reference card for information on the instructions. The screenshot of the GUI is shown in Figure 2.

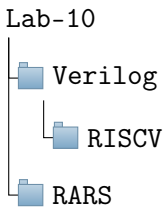
Verilog and the Behavioral Simulator: Verilog is a Hardware Description Language (HDL) that is used to model hardware. Verilog is a lot like C, except variables are in terms of *"registers"* (which are just flip-flops) and components are organized into modules which are connected by wires – just like a real circuit. A behavioral simulation models the behavior of the circuit, without worrying about the way the circuit is implemented. For example, you can declare a for loop (with a static number of iterations) – this would ultimately be converted to a state machine in hardware, but for the purpose of simulation, it is just a for loop. Another important note is that, unlike a conventional program where instructions execute sequentially, events in hardware happen in parallel. There are MANY simulators for Verilog – some are standalone, and some are built into other tools. For this lab, we will use ISim, a hardware description language (HDL) simulator used for functional and timing simulation of Xilinx FPGAs. You need to know how to simulate an HDL code (such as Verilog) using ISim.

Reading Check

1. What is the trigger and payload in hardware Trojan?
2. What are the differences between *combinational* and *sequential* Trojan? ?
3. What are the roles of registers in the CPU?

D Getting Started

Download and open the Lab-10 directory. The directory is structured as follows:



The RISCV folder contains a simple RISCV CPU implementation in Verilog. The filenames should be quite familiar from computer architecture (IF, ID, EX, MEM, WB...). Notice there is an instruction memory file (inst_mem.v) that implements a simple memory module. One particular line (initial $readmemb ("instruction.txt", inst_memory);) instructs the simulator to load a text file called *instruction.txt* at the start of the simulation (the "initial" keyword) into that module. You can use the assembly code generated by the RARS simulator to run the behavioral simulation in Verilog –simply dump the assembled code to file and copy/paste the contents into the instruction.txt file in ISim. See attached manual for more details.

E Lab Assignment

Your task in this assignment will be implementing a Hardware Trojan in the processor, which a specific memory access pattern can activate. Eventually, you will need to demonstrate the activation and the effect of your designed Trojan in the processor.

1. Design your Trojans, using the design principles discussed in class. You are required to create one combinational Trojan, and one sequential Trojan. The Trojans should be *stealthy*:
   - Do not add any additional input/output pins to any of the modules.
   - Do not make the trigger too common – for example, your Trojan should not trigger every time there is an add instruction – that would be caught right away!
   - The Trojan should be small – minimize the amount of additional logic needed to make your Trojan work as desired.

   The last point (Trojan size) cannot be directly assessed from simulation, but it can be gauged at a high level: for example, a trigger that requires the ANDing of 30 different inputs will definitely be larger than one that triggers by ANDing 5 different inputs. A sequential Trojan that requires 8 states prior to triggering will need 3 flip flops + associated state transition logic, whereas one that requires 4 states needs 2 flip flops + state transition logic. It would also have a lower impact on power but may trigger more frequently. Keep these trade-offs in mind as you design your Trojans.
2. Demonstrate the effect of the Trojan by displaying the mismatched register values comparing the RARS and ISim simulator. Include similar screenshots clearly annotated with the expected (Trojan-free) operation and faulty (Trojan-included) operation. Also include a screenshot of the waveform showing the Trojan triggering for both the sequential and combinational Trojans. Figure 3 shows an example output for a simple Trojan that triggers when a particular register value is used in the datapath. The inserted Trojan has been designed to trigger when the value 0x777 is input as operand1 into the ALU. This input triggers the payload which forces all 0s on the output, rather than the correct output for that input and operation. Notice that when operand1=0x777, the reg values of x30 and x7 show normal execution in the case of a Trojan-free design, but the reg values display all 0s with the Trojan inserted in the design.

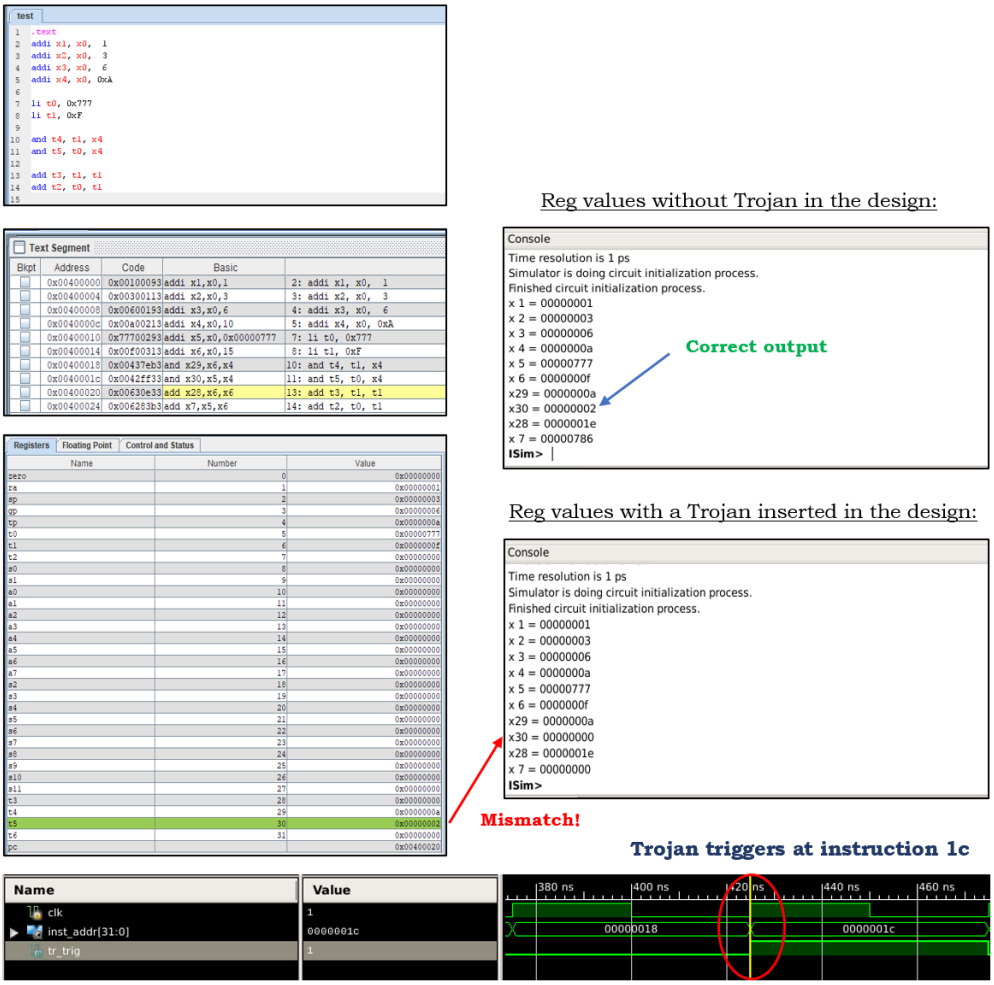


Figure 3: An example output showing the expected reg values for Trojan-free and Trojan-inserted design from RARS and ISim.

F Badges and Achievements

This experiment involves the use of **HDL**, **RISC-V**, and **Assembly**. By successfully completing the lab you can unlock the following achievements:

- HDL3: Verilog HDL
- FPGA5: FPGA Programming and Interfacing
- COM4: Advanced Serial UART

Depending on the demonstrated skill, each achievement will be further divided into bronze, silver, and gold levels.

G Deliverables

Follow all report guidelines as detailed in the manual. The following result(s) must be included in the report:

- Verilog designs contain a combinational and a sequential Trojan.
- Assembly codes that trigger the combinational and sequential Trojans.
- Screenshots showing a) Register values (in RARS) and register contents (for Trojan-free and Trojan-inserted designs) showing correct/expected output matching the RARS window, and mismatched output for the Trojan-inserted design; b) The waveform output from ISim showing the trigger signal going high to enable the Trojan. All files should be clearly annotated, indicating areas of interest.

Submit your report along with all code. All analysis code should be runnable and should produce the output submitted with your report.