CMPE-361 Intro to Hardware Security

Project 1

Final Paper

By submitting this report, you attest that you neither have given nor have received any unwarranted assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Barak Binyamin
Performed November 3rd, 2023
Submitted November 6th, 2023

Lab Section 01L1
Instructor: Dr. Michael Zuzak, Ph.D
TA: Long Lam, Colin Vo, Sydale John Ayi
Lecture Section 0
Professor: Dr. Michael Zuzak, Ph.D

Introduction

This exercise explored the development of a methodology for hardware trojan detection on an FPGA. First, it was demonstrated that bitstreams with simple hardware trojans could be differentiated by output to pseudo randomly generated inputs. Lastly, it was demonstrated that information about trojan type and functionality could be loosely derived through further investigation of the input and output of trojan infected bitstreams.

Theory

Hardware trojans are malicious modifications to integrated circuits that enable disruption, degradation, or unauthorized access [1]. There are a few types of hardware trojans, including **combinational** trojans, which are triggered on the occurrence of logic values (usually rare internal nodes), and **sequential** trojans, which exhibit malicious effects after a sequence of logic values [2]. Some hardware trojans affect the output of a device, while others make discrete changes that may make a side-channel attack more approachable, or decrease the intended life of a product, for example by thinning a wire. Through the examples provided, this exercise explores the detection of hardware trojans that have a measurable effect on output.

The problem is to develop a methodology to identify hardware trojans in FPGAs when testing an arbitrary design. Following the **STRIDE** methodology, it is crucial to identify **assets**, **possible adversaries**, **features that enhance or weaken a design**, as well as some **threats**. It is also crucial to identify the severity of each **vulnerability**, this can be done using the **DREAD+D** approach. Assets include functionality, usage statistics, and sensitive information embedded or passing through the device. One large **adversary** is an **untrusted foundry**. Untrusted foundries the ability to tamper with the design, adding or changing elements of the device to fit their goals. The threat is **tampering** while the vulnerability is the **integrity** of the device. Assume the DREAD+D (Damage, Reproducibility, Exploitability, Affected Users, Discoverability, Detection) categories were each graded on a 1-5 scale, 1 being low, 5 being high severity. The damage from an unfunctional device can be quite severe, especially in mission critical environments, for that reason the threat of a kill switch or any device that hinders functionality should be rated high, 5/5. A foundry has the ability to easily reproduce trojan hardware, so this category should be rated 5/5. Every user of the device with a trojan is potentially affected, so the Affected Users category should earn a 5/5. Effective and well designed hardware trojans require many resources, including an understanding of a design to make well informed decisions on trojan placement. For this reason, Discoverability would earn a 2/5. Some trojans are easier to find than others, but some trojans are cleverly designed (hence the term trojan) so detection is difficult, earning this category 5/5. That concludes the categories of DREAD+D, making the threat level of this analysis 22/25, a HIGH threat level. Untrusted foundries have access to a large swath of resources and represent a worthy adversary due to their ability to embed hardware trojans into commissioned devices. It is crucial to identify inexpensive and effective methods of detecting these hardware trojans, as their impact may be unrecoverable.

Depending on its sensitivity, infected hardware can be differentiated from safe hardware by comparing their outputs. Infected hardware will sometimes produce different outputs then safe hardware. These known to be safe references for hardware are often referred to as "**golden**". It is often that the input space is too large to compare all possible combinations in a reasonable amount of time. There are a few options to sample a device. Taking the first N inputs is an option, but it may lack coverage on the top few bits. Pseudorandom generated inputs can allow for a more even space of coverage [3]. The worst case probability of finding a single input combinational trigger can be seen in Equation 1.

$$f(S) = \frac{1}{N^2} \times S \times 100 \quad (1)$$

In Equation 1, N represents the number of bits in the input, while S represents the number of random samples. Using Equation 1, a test of hardware with 32 input bits, and 1000 samples would have returned a 0.97% chance of finding a trigger. Thankfully, the odds increase significantly when the trigger becomes more sensitive. A trigger can increase its sensitivity by triggering on more inputs, or triggering on looser logic (logic that produces more truthy responses, for example OR produces more truthy outputs than AND). The probability of finding a sequential trojan can be far worse due the fact that a finite but large amount of previous states could be factored into the input space of the trigger. The worst case probability of finding a single input combinational trigger can be seen in Equation 2.

$$f(S) = \frac{1}{M \times N^2} \times S \times 100 \quad (2)$$

In Equation 2, N represents the number of bits in the input, M represents the number states that can be stored, while S represents the number of random samples.

The next step after discovering that a trojan exists, is characterizing it. This can be approached by analyzing the inputs that resulted in unexpected outputs. Examining the number of occurrences that a 1 resides in bit positions can show a pattern if the trigger is combinational. If 1's occur a significant amount more or less than in other bit positions, it is likely that the bit position is involved in the trigger. This approach can be applied to sequential trojans as well by factoring in N previous outputs as additional inputs. The automation scripts mentioned in the Measurement Methods section only factored in N=1 previous outputs as an input.

The percent of the inputs that caused unexpected outputs can give insight into how sensitive the trigger is. As discussed earlier, triggers are more reactive when they have looser logic or more sensitive to a larger number of inputs.

Measurement Methods

This exercise utilized a Basys 3 trainer board. Included on the board is JTAG programming hardware and an Artix-7 FPGA. The board has community support and official support offered by Digilent. Each bitstream provided a serial wrapper to interface with the target's inputs and outputs. OpenOCD software was used to automate the upload of bitstreams to the FPGA, while a test suite utilizing python, make, & bash was implemented to sample inputs, store outputs to JSON files, and run some simple analysis [4]. The references to these scripts and tools can be found in Appendix 1.

Results & Analysis

The automation scripts were run on training, test, and challenge bitstreams. Training bitstreams were used to improve detection as the trojan Verilog was provided for each, while test bitstreams provided a way to evaluate the methodology on unknown targets (two bitstreams were unlabeled, a third golden was generated given the Verilog base and a serial wrapper). Challenge bitstreams included a trojan made to be more challenging to detect and characterize. A reference to the bitstreams, as well as output logs, can be found in the git repository mentioned in Appendix 1.

The first training bitstream was labeled c432. The input to this design is 36 bits, while the output is 7 bits. The trojan functionality was characterized by output[1] = output[1] XOR 0x01. The trigger mechanism was characterized by NOT input[0] AND input[16] AND input[30], which is combinational. The automation script was able to successfully identify the bitstream as a trojan, indicating the affected bits, but could not indicate the correct trigger bits.

The second training bitstream was labeled c5315. The input to this design is 178 bits, while the output is 123 bits. The trojan functionality was that another bus (YBUS) is used instead of (XBUS) when the trigger is activated, possibly exposing internal information through the output. The trigger mechanism was characterized by input[8] AND NOT input[10] AND input[12] AND NOT input[14], which is combinational. The automation script was able to successfully identify the bitstream as a trojan, indicate the affected bits, and indicate some correct trigger bits.

The third training bitstream was labeled c6288. The input to this design is 32 bits, while the output is 32 bits. The trojan functionality was characterized by output[28] = output[29] XOR output[28] XOR output[27]. The trigger mechanism was characterized by NOT input[29] AND NOT input[25] AND NOT input[6] AND input[2] AND input[13], which is combinational. The automation script was able to successfully identify the bitstream as a trojan, indicate the affected bits, and indicate all correct trigger bits.

The first test bitstream was labeled c499. The input to this design is 41 bits, while the output is 32 bits. The automated test script was able to identify that the first of the two unknown bitstreams had a trojan. The trojan functionality was characterized by output[15] = ?. The most educated guess by comparing the number of affected bits, the percent of affected outputs, and number of projected combinational and sequential bits, is that the tigger is likely combinational.

The second test bitstream was labeled c1908. The input to this design is 33 bits, while the output is 25 bits. The automated test script was able to identify that the second of the two unknown bitstreams had a trojan. The trojan functionality was characterized by output[11] = ?, output[10]=?, output[9]=?, output[8], output[7]=?, output[6]=?, output[5]=? , and output[4]=?. The most educated guess by

comparing the number of affected bits, the percent of affected outputs, and number of projected combinational and sequential bits, is that the tigger mechanism is likely combinational, involving 1 or more XOR's of the input bits 14, 6, 5, 4, 3, 2, 1, 0.

The bitstream labeled FIR was the only challenge bitstream attempted. The input to this design is 32 bits, while the output is 32 bits. The automation script was able to successfully identify the bitstream as a trojan. The trojan affected all output bits in an unknown way. The most educated guess by comparing the number of affected bits, the percent of affected outputs, and number of projected combinational and sequential bits, is that the trigger is likely sequential involving the first previous output bits 16, 1, 0 as well as input bit 11.

Discussion and Conclusions

Detecting and characterizing even simple hardware trojans proved to be a difficult task. Some challenges included improving golden sampling time, data formatting, data manipulation, implementing XOR and XOR properly in python, bitwise operations in python, managing MSB-LSB (sending and receiving the input/output bits in the correct order), serial input/output padding (managing that the type of serial communication was limited to sending bytes not bits), and serial desynchronization.

The trojan detection and characterization methodology focused more on combinational trojans as it did not does not repeat input values intentionally, and searched more heavily for signs of combinational triggers. Some of the detection methods were not fine tuned enough to consistently identify combinational trigger bits, and the methodology was not fully automated and not design focused, as it did not take the netlist into consideration, for example to analyze rare nodes.

The detection methodology proved to be successful in identifying all trojan bitstreams and characterizing some key aspects of the trojans provided.

There are many ways to improve and expand upon this trojan detection and characterization methodology. One of which would include the analysis of rarely activated nodes using an Automated Test Pattern Generation tool and SAT solver called *Atlanta* [5]. Atlanta could also be used to create inputs that could isolate specific bottlenecks in a design. Given knowledge of bottlenecks this isolation could be used to identify which portion of a circuit, and which input bits, might be affected by the trojan. Another improvement vector would be to involve some decision tree to narrow down the input bits possibly involved, while incorporating a live comparison of input output values to the golden. A last and major undertaking would be to create a machine learning model to generate hardware trojans and a machine learning model to detect hardware trojans. The machine learning model to generate hardware trojans could be trained off a data set of existing hardware trojans. The machine learning model to identify hardware trojans could be trained off the new hardware trojans that the generation model provides [6,7,8].

Appendix

Appendix 1: Tools for Automated Upload and Test

The git repository listed at https://github.com/BarakBinyamin/Trojan-Detection contains full usage documentation, output logs, automation scripts, and bitfiles ready for testing. The included programs and dependencies are all compatible with windows, mac, and linux.

References

[1] Dr. Michael Zuzak, *CMPE361_Project1 Description*, https://mycourses.rit.edu/d2l/le/content/1044747/viewContent/9493257/View
[2] Xinmu Wang, *Sequential Hardware Trojan: Side-channel Aware Design and Placement*, https://swarup.ece.ufl.edu/papers/IC/IC1.pdf
[3] Python, *Python Random Module*, https://docs.python.org/3/library/random.html
[4] openOCD, *Open On-Chip Debugger*, https://openocd.org/
[5] Virginia Tech, *Atlanta ATPG*, https://github.com/hsluoyz/Atalanta
[6] ACM Transactions on Embedded Computing Systems, *Hardware Trojan Detection Using Machine Learning*, https://dl.acm.org/doi/full/10.1145/3579823
[7] Yu, Shichao, *Automated hardware trojan detection*, https://pureadmin.qub.ac.uk/ws/portalfiles/portal/251958201/Thesis_ShichaoYu.pdf
[8] Hassan Salmani, *Reference-free Hardware Trojan Detection in Gate-level Netlist (COTD)*, https://apps.dtic.mil/sti/tr/pdf/AD1041368.pdf