

# Defeating Hardware Trojan through Software Obfuscation

Andrea Marcelli, Marco Restifo, Ernesto Sanchez, Giovanni Squillero  
Politecnico di Torino

Corso Duca degli Abruzzi 24, 10129, Torino

{andrea.marcelli, marco.restifo, ernesto.sanchez, giovanni.squillero}@polito.it

**Abstract**—In recent years a new kind of threat, known as Hardware Trojan, is affecting the Integrated Circuit industry. Due to the segmentation in the production, untrusted parties involved in the supply chain may illegally inject additional hardware components, that, under specific circumstances, act for malicious purposes. While it is mostly unfeasible to identify malicious hardware tampering with in-lab testing, as a remedy, several countermeasures have been proposed, mostly based on hardware alterations of the original design, with the main drawbacks of increased production costs, increased area and energy consumption. In this paper, we introduce a cost effective solution, completely software-based, that minimize the chance of activation of a multi-stage trigger Hardware Trojan. The proposed approach relies on a software obfuscation mechanism, which exploits evolutionary algorithms to modify an executable program without affecting its original functionalities. Such always-changing, obfuscation routine, can be used to protect critical infrastructures and operations, at a minimum and predictable loss of performances. To show the effectiveness of the proposed technique, we developed a proof-of-concept evolutionary obfuscator and we are going to test it against a well-known real-world hardware attack scenario.

## I. INTRODUCTION

The current trend of Integrated Circuits (ICs) of moving to smaller transistors increased the overall hardware performances, while on the other hand dramatically raised the production costs. As a consequence, most of the semi-conductor companies prefer to focus on the circuit design only and outsource to third-parties the fabrication phase. This radical change raises concerns about the trustiness of all the parties involved in the production-chain and the threat of a hardware piracy, through the insertion of a Hardware Trojan Horse (HT) [1], [2], became reality. In order to guard against any type of production error, both intentional and not, companies heavily rely on post-production logic testing. However, since most of the malicious hardware alterations are usually activated by a very low reproducible sequence of events, it is very unlikely that traditional test strategies trigger and detect them.

According to a model presented in [3], the HT conceals its malicious behavior during normal operations: it monitors specific circuit inputs or internal signals and activates the *payload* only when a predefined pattern, known as *trigger*, is detected. However, to decrease the HT chance of being activated during logic testing, usually the trigger is connected

to signals with very low controllability, and the payload is attached to signals with very low observability.

The security research community proposed several techniques to discover hidden HT [2], such as functional tests [4], runtime monitoring [5]–[8], and side-channel based approaches [9]–[11]. However, the effectiveness of existing methods is subject to a number of practical challenges. For example, in side-channel based approaches, such as power or frequency analysis, the measured power values could vary significantly due to many natural and non-malicious factors, resulting in high false positives/negatives in the detection results. Furthermore, the physical inspection approach would introduce an extremely high cost to the IC supply chain despite its effectiveness [12], [13].

In 2016 Yang et al. [14] presented an interesting fabrication-time attack that could lead to a controlled privilege escalation attack. By injecting a single gate and a capacitor in the open spaces of an already placed and routed design, Yang et al. were able to force the flip-flop that holds the privilege bit to a desired value.

### A. Threat scenario

Due to the impossibility of verifying any piece of hardware to guarantee its trustworthiness, we conservatively consider any hardware platform as *untrustable* and already *compromised* by malicious parties, according to the model presented in [3]. On the other hand, it is important to remind that the hardware itself only represents the necessary underlying infrastructure to carry out an attack, but it is harmless without a specially crafted software that activates the malicious trigger function. In the same way, we define any software that could be executed on a particular hardware platform as *untrustable* and already *compromised* too.

### B. Our contribution

In order to overcome these limitations, in this paper we propose an innovative approach that completely relies on a software solution that exploits an evolutionary algorithm to generate a new obfuscated version of the code to be executed. In a previous work [15], we introduced the idea of an evolutionary opcode generator to hide the core functionalities of an always-mutating malicious software, or malware, to challenge antivirus products. In this research, we brought a similar

idea to the software-obfuscation of critical code to avoid the triggering of malicious hardware injected components.

To demonstrate the effectiveness of the proposed method, we deployed a proof-of-concept evolutionary obfuscator to challenge the multi-stage trigger introduced by Yang et al. [14]. Preliminary results show that the software-obfuscation causes the failure of most of the attacks at a minimal loss of runtime performances.

The rest of the paper is organized as follows. Section II introduces the proposed approach. Section III shows the current experimental setup, while Section IV discuss the limitations of the solution. Section V concludes the paper.

## II. PROPOSED FRAMEWORK

In this paper, we propose an evolutionary-based solution to mitigate the possible activation of a multi-stage trigger HT inserted in a processor core, by modifying the processor software. In particular, the proposed method is able to slightly modify the software in order to eliminate the sequence of instructions that may activate a HT possibly inserted in the targeted processor. According to the model we developed, each software that needs to be executed on a particular hardware platform, must be preprocessed by the evolutionary obfuscator.

The core of the software obfuscator is a Turing-complete evolutionary algorithm able to generate completely new variant of the original software mostly without affecting its functionality. All the code alterations are performed at the program assembly level and later simply converted to a binary file, so no high-level compiling and linking phase is needed.

Since even a few instructions (i.e., a sequence of 3 or 4 instructions) may trigger the activation of HT, it is necessary to insert, substitute or partially shuffle some assembly instructions in the original code, although this has to be done carefully to prevent the potentially modifying of the program functionality. In detail, the proposed obfuscation process enables four types of possible mutations of the original code.

The first code mutation simply adds a new no-operation (NOP) instruction, while the second inserts a couple of meaningless instructions before or after the selected position. The third one swaps the current instruction with the one above or below it. Finally, the last mutation substitutes a candidate instruction with one or a set of logically equivalent, i.e. the new instructions have the same output than the original one. Clearly, this mutation requires a deeper analysis of the program flow in order to do not modify the source program functionality.

In order to efficiently evaluate a candidate solution, the Jaccard Similarity Index, the simulation time, and the number of memory accesses are used to assess the candidate fitness value. Aiming to destroy the trigger capabilities through program diversity, the process is iterated for a given number of generations, or until the Jaccard Index is lower than an experimentally-defined threshold.

Since the application of the software obfuscation introduces a minimum but predictable loss of performances, the run time

execution may be affected by the new instruction substitution that could not be optimized for the task under execution. However, as far as the introduced delay is known, it could be added as a constraint during the evolutionary process.

## III. EXPERIMENTAL SETUP

To demonstrate the effectiveness of the proposed method, we implemented several variants of a multi-stage trigger Hardware Trojan in a MIPS-like processor core that is freely available at [16]. The choice was dictated by the similarity with the processor cores commonly available in the embedded market.

The MIPS-like 32-bit RISC processor core includes a 5-stage pipeline with hazard detection, a pipeline interlock, a branch prediction unit and a system co-processor. The synthesized version of the processor counts about 46k equivalent gates, while the instruction set is made of 52 instructions, which includes load and store, arithmetic, logic, and branch related instructions.

As a testbed, we selected a set of benchmark programs that may suit the characteristics of typical software developed for embedded systems. In the following, the experiments performed on an implementation of the Fibonacci algorithm, and matrices multiplication, are analyzed in details. Additional benchmark programs based on MiBench will be presented on future works.

In order to emulate the behavior of a HT, we consider each program has been injected a three-stage HT trigger sequence. Although the actual sequence of three instructions that activates the HT is unknown, assuming that the initial code contains only one raising triplet of instructions, at the end of the process we evaluate the drop in the probability of activating the malicious behavior.

The implementation of the unrolled version of Fibonacci takes two input numbers, owing to the Fibonacci series, and generates the next 8 values. The original version of the code contained 28 triplets of instructions, while in the final code, only 1 of those was left. Assuming the existence of an offending sequence among the 28 ones, we can estimate the drop of probability of the HT activation about 96.4%. Figure 1 shows the Jaccard index evolution over generations. As illustrated in Figure 2 and 3, the execution time and memory access are not affected by the code obfuscation process.

In the a second test, two square matrices are given in input and the program computes their product. While the code originally contained 67 triplets, in the new modified version only 15 of those are left. Results shows that the drop of probability in HT activation is about 77.6%. Figure 4 illustrates the Jaccard index evolution over generations, while Figure 5 and 6 show that obfuscation increases execution time about 47.7% and memory access about 21.1%.

## IV. LIMITATIONS

Similarly to other security solutions that rely on a probabilistic approaches, the proposed method is far to be completely reliable. While the proposed evolutionary obfuscator

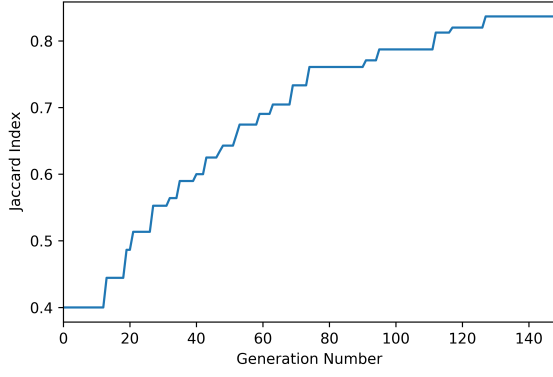


Fig. 1. Jaccard Index Fibonacci

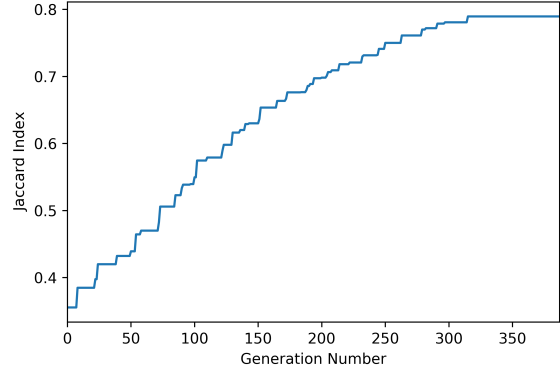


Fig. 4. Jaccard Index matrix product

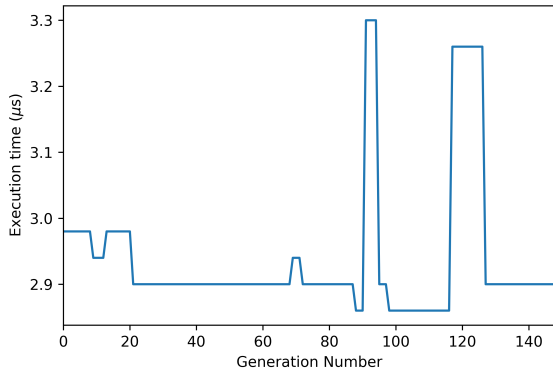


Fig. 2. Execution Time Fibonacci

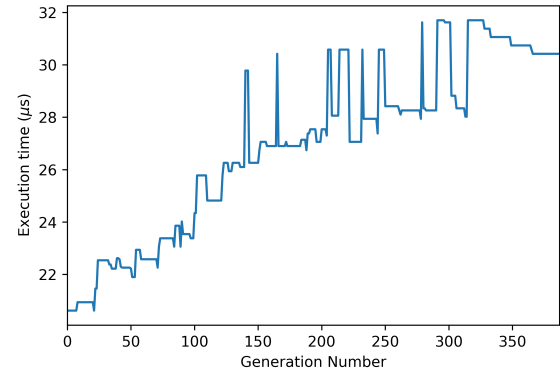


Fig. 5. Execution Time matrix product

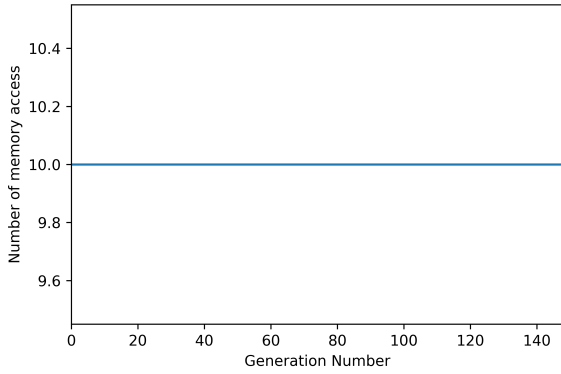


Fig. 3. Memory Accesses Fibonacci

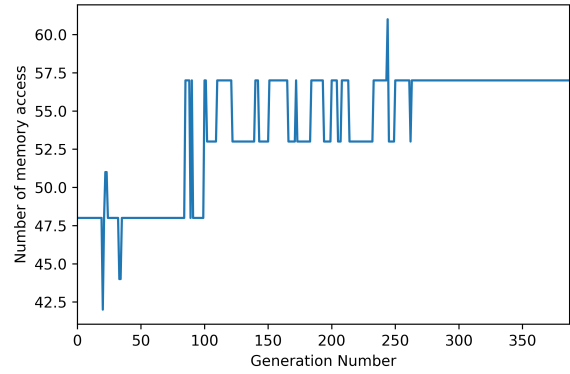


Fig. 6. Memory Accesses matrix product

extremely minimize the possibility of triggering malicious hardware components, there exist a rare possibility that, after the obfuscation process, the new assembly instructions will still trigger the activation of the inserted Hardware Trojan. Moreover, it is theoretically feasible that a clean software could be transformed into a malicious one, if the new sequence of instructions perfectly coincide with the triggering sequence.

Although both scenarios exist in theory, we are firmly confident that in practice these are so rare, to be considered negligible. In addition, several attacks require a specific timing sequence, that is, instructions to be executed after the hardware payload has been activated. As an example, in the attack described by Yang et al. [14], the change of the privilege mode should be treated accordingly, in order to access the privileged

processor state.

Finally, it should be noticed that any hardware alteration, both intentional or not, causes an irremediable modification to the hardware structure. Consequently, the original design is forever compromised, resulting in an unpredictable behavior if specific triggering conditions are met. Neither the software obfuscator, nor any other software-based solution, could ever overtake the limitations of a non-proper-working hardware.

## V. CONCLUSION

In this paper, we introduced a novel software obfuscation technique that effectively increase the security of a system in the presence of a hardware affected by malicious alterations. Among the existing Hardware Trojan variants, previous researches showed how the multi-stage trigger currently represents the most hiding, hence powerful, type of attack. Exploiting a pure software evolutionary approach, we are able to prevent, at a minimum and predictable loss of performances, such a helpless scenario.

In order to test the effectiveness of the introduced methodology, we selected some real-world attacks and developed a proof-of-concept of the software obfuscator. While detailed experiments are still under analysis, preliminary results show that the evolutionary obfuscator was successful in preventing most of the privilege escalation attempts.

## ACKNOWLEDGMENT

Andrea Marcelli's Ph.D. program at Politecnico di Torino is supported by a fellowship from TIM (Telecom Italia Group).

## REFERENCES

- [1] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. IEEE, 2008, pp. 15–19.
- [2] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.
- [3] S. Dupuis, G. Di Natale, M.-L. Flottes, and B. Rouzeyre, "Identification of hardware trojans triggering signals," in *First Workshop on Trustworthy Manufacturing and Utilization of Secure Devices*, 2013.
- [4] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards trojan-free trusted ics: Problem analysis and detection scheme," in *Proceedings of the conference on Design, automation and test in Europe*. ACM, 2008, pp. 1362–1365.
- [5] G. Bloom, B. Narahari, and R. Simha, "Os support for detecting trojan circuit attacks," in *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*. IEEE, 2009, pp. 100–103.
- [6] J. Dubeuf, D. Hély, and R. Karri, "Run-time detection of hardware trojans: The processor protection unit," in *Test Symposium (ETS), 2013 18th IEEE European*. IEEE, 2013, pp. 1–6.
- [7] X. Zhang, A. Ferraiuolo, and M. Tehranipoor, "Detection of trojans using a combined ring oscillator network and off-chip transient power analysis," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 9, no. 3, p. 25, 2013.
- [8] A. Ferraiuolo, X. Zhang, and M. Tehranipoor, "Experimental analysis of a ring oscillator network for hardware trojan detection in a 90nm asic," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2012, pp. 37–42.
- [9] M. Banga and M. S. Hsiao, "A region based approach for the identification of hardware trojans," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. IEEE, 2008, pp. 40–47.
- [10] S. Wei and M. Potkonjak, "Self-consistency and consistency-based detection and diagnosis of malicious circuitry," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 1845–1853, 2014.
- [11] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. IEEE, 2008, pp. 51–57.
- [12] J. M. Soden, R. E. Anderson, and C. L. Henderson, "Ic failure analysis: Magic, mystery, and science," *IEEE Design & Test of Computers*, vol. 14, no. 3, pp. 59–69, 1997.
- [13] N. Hu, M. Ye, and S. Wei, "Surviving information leakage hardware trojan attacks using hardware isolation," *IEEE Transactions on Emerging Topics in Computing*, 2017.
- [14] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog malicious hardware," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 18–37.
- [15] M. Gaudesi, A. Marcelli, E. Sanchez, G. Squillero, and A. Tonda, "Challenging anti-virus through evolutionary malware obfuscation," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 149–162.
- [16] "minimips," <http://opencores.org/project,minimips>, accessed: 15-Sep-2016.