Towards Practical Fabrication Stage Attacks Using Interrupt-Resilient Hardware Trojans

Athanasios Moschos
Georgia Institute of Technology
Atlanta, Georgia, USA
amoschos@gatech.edu

Fabian Monrose
Georgia Institute of Technology
Atlanta, Georgia, USA
fabian@ece.gatech.edu

Angelos D. Keromytis
Georgia Institute of Technology
Atlanta, Georgia, USA
angelos@gatech.edu

***Abstract*—We introduce a new type of hardware trojans called *interrupt-resilient trojans* (IRTs). Our work is motivated by the observation that hardware trojan attacks on CPUs, even under favorable attack scenarios (*e.g.,* an attacker with local system access), are affected by unpredictability due to non-deterministic context switching events. These events can lead to race conditions between trigger signals and the CPU events targeted by the trojan payloads (*i.e.,* a memory access), thus affecting the reliability of the attacks. This work shows that interrupt-resilient trojans can guarantee the reliable implementation of sophisticated trojan attacks through punctual delivery of the triggering signal under diverse context-switching conditions. We successfully utilize IRTs in different attack scenarios against a Linux-capable CPU design and showcase their resilience against context-switching events. More importantly, we show that IRT designs allow for seamless integration during fabrication stage attacks. We evaluate interrupt-resilient designs on a high-speed, tape-out ready RISC-V layout in a 28nm commercial technology process, adding an average overhead delay of only 20 picoseconds, while leaving the sign-off characteristics of the layout intact. In doing so, we challenge the common wisdom regarding the low flexibility of late supply chain stages (*e.g.,* fabrication) for the insertion of powerful trojans. To promote further research on microprocessor trojans, we open-source IRTs and their supporting software logic.**

***Index Terms*—Hardware Trojans, Computer Architecture, Very Large Scale Integration, Integrated Circuits, RISC-V**

I. Introduction

Hardware trojans (HTs) have become a topic of increased attention, due to the covertness of their nature and their potential for malicious exploitation of globalized supply chains. In their most clandestine form, hardware trojans are forged by intentional modifications made directly to the physical layout of integrated circuits (ICs). Their stealthy nature has sparked great interest in the offensive hardware security domain [18], with existing research shedding light on the attackers' powers at different stages of the chips' design cycle. Indeed, the large body of offensive research [3, 4, 10, 11, 17, 19] has advanced our collective understanding of different hardware trojan design capabilities, culminating in new lines of inquiry in the implementation of practical trojan attacks [3, 4, 19].

To date, much of that research [18] has focused on the stealthiness aspect of trojan attacks, where the number of trojan gates and nets partaking in the malicious modification is the key metric used to characterize a trojan's stealthiness [7, 11, 17, 19]. Existing literature on trojan attacks against CPUs [3, 4, 6, 10] often assumes threat models with attackers that are able to execute malicious binaries and surreptitiously excite trojans hidden in the microarchitecture to deliver their payloads. An often overlooked factor however, is how the interplay between software and hardware in modern microprocessor systems can pose challenges for the correct execution of a hardware trojan attack. Specifically, modern CPUs are complex finite state machines that handle numerous unpredictable, asynchronous events (*e.g.,* interrupts). Thus, it is not rare for a CPU state change to happen between the trojan triggering and the delivery of the effect. If that change is not properly accommodated for, it can lead to undesirable side effects (*i.e.,* a crash). Occurrence of unexplained system behaviors can raise attention and lead to further device scrutiny. *We therefore consider attack reliability to be as equally as important as a trojan's size when it comes to stealthiness.*

Despite existing efforts on CPU-based trojan designs [7, 11, 17, 19], there has been a lack of research on how the subtleties of asynchronous events can undermine attack stealthiness. Motivated by this observation, we shed light on the spectrum of context switching (CS) events that can occur during HT attacks against CPUs. Moreover, from the stand-point of a fabrication stage attacker, we examine two CS-resilient trojan designs that can be efficiently implemented in tape-out ready layouts. We make the following contributions:

- We introduce *interrupt-resilient trojans* (IRTs), that tackle context-switching events and manage the reliable delivery of the time-critical trigger signal.
- We show IRTs' attack performance against CPUs under different context switching scenarios.
- We identify components in modern microarchitectures that can host IRTs during fabrication stage attacks.
- We emulate a fabrication stage attacker and insert IRTs in a tape-out ready CPU layout.

Overall, we bring attention to practical challenges in hardware trojan attack design and revisit longstanding assumptions about the lack of flexibility in fabrication stage attacks [9].

II. Background and Related Work

In computer security parlance, an HT consists of a trigger and a payload. The *trigger circuit* is tasked with monitoring for predefined conditions and initiating the payload delivery when these conditions are met. Trigger circuits that are easy to excite at the attacker's behest but also remain hidden during

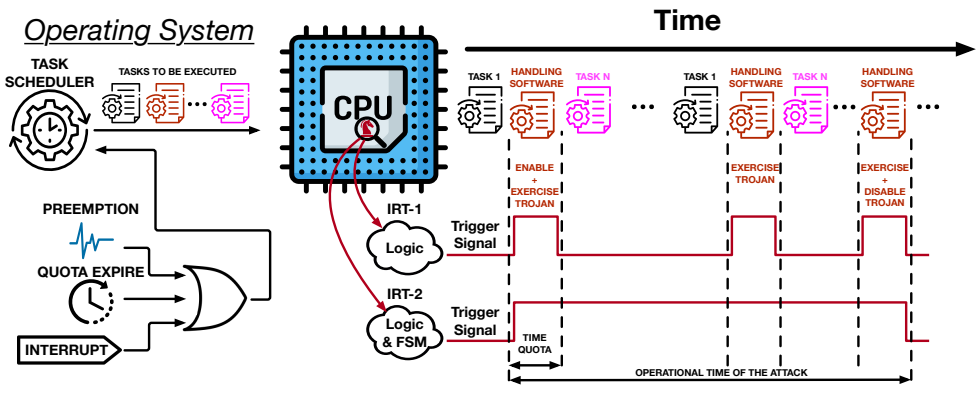


Fig. 1: Time slicing of the CPU's execution time during a hardware trojan attack.

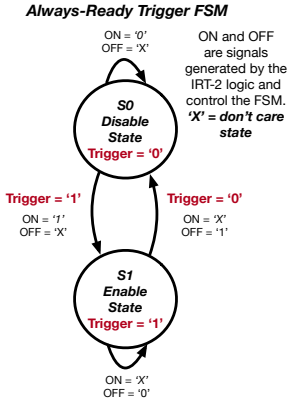


Fig. 2: The IRT-2 FSM.

the testing phase or normal chip operation, are considered highly reliable. The second vital component, the *payload circuit*, is responsible for modifying the host system's native behavior. However, a trojan is not an organic part of the CPU's design and only stages prior to fabrication [9] provide smooth microarchitectural integration, whether through high-level specification adjustments or low-level RTL logic modifications at the more lenient front-end and back-end stages. Central to hardware trojan attacks is the effective but covert communication with the trojan, through a *handling software*. Discontinuities in the software and hardware collaboration cycle can make a trojan more easily discoverable.

Contemporary Attacks: State of the art research on CPU HT attacks covers different insertion stages (design vs fabrication) and access levels (local vs remote). For the most part, the size of the HT implementations — namely, the number of trojan gates and signals that are necessary to implement the trojan — is often used to justify arguments about stealthiness. Unfortunately, the operational aspect of a HT is usually considered orthogonal to its stealthiness and often considered out of the scope of [7, 8, 14], which lack a discussion about the conditions under which their HTs operate. For the studies that do consider operational issues (*e.g.,* [3, 4, 6, 11, 13, 19]), they usually lack an in-depth discussion of how the intricacies of the CPU's operational reality can undermine the effectiveness of their proposed attacks. Indeed, several approaches simply assume that attackers are readily able to execute malicious code on the target *without interruption*.

Challenges in Executing a Hardware Trojan Attack: Modern OSes segregate virtual memory into user space (*e.g.,* software applications) and kernel space (*e.g.,* privileged OS kernel, device drivers) in order to provide memory and hardware protection from malicious or erroneous software behavior. Furthermore, to improve performance, modern microprocessors interface with peripheral modules and allow for OS multitasking. The latter is accomplished through the operation of context switching, where the state of a process or thread is saved on interrupt and then later restored once its execution is resumed. Multitasking, interrupt handling, user and kernel mode switching are the primal reasons behind context switching events. Undoubtedly, the assumption that attacks will execute without interruption is unrealistic.

Design stage attacks allow for a smooth HT integration with the microarchitecture, avoiding the challenges of context switching. Thus far, only De et al. [3] have even acknowledged the challenges posed by context switching during a hardware trojan attack, and Tsoutsos and Maniatakos [17] have assumed the presence of context switching events. Specifically, they show how CS events are leveraged by the HT to provide to a malicious user process access to unauthorized memory. However, their implementation considers a specific attack without guarantees on its correct execution.

Regardless of the assumptions made regarding the attacker's access vector on the system, the handling software must deal with unpredictable interruptions from context switching. It is precisely this uncertainty that we aim to tackle head on. Specifically, we argue that *any realistic hardware trojan attack must accommodate for non-deterministic context switching events*, generated by the continuous interleaving of interrupts and applications in the execution chain of a microprocessor.

III. Threat Model

Within the security community, fabrication stage attacks are considered to be particularly restrictive [3, 9, 17, 18, 19] in terms of the attackers' powers (*e.g.,* limited design information, rigidness in sign-off layout modifications). We question this longstanding assumption, and instead show that sophisticated HTs can indeed be implemented efficiently at this phase and lead to attacks equally as powerful as those enabled by insertion of HTs in design phases prior to fabrication [9].

Specifically, we adopt the threat model of a fabrication-stage attack, where a malicious entity inside a foundry gains access to the chip's GDSII file (*e.g.,* the input file for fabrication) to introduce an IRT. We assume that the design process up to the GDSII generation is completely trusted. The malicious alterations take place inside the foundry by an attacker with access to necessary tools (e.g., design automation software, process design kit, cells and libraries of the victim layout). Modern chips have areas occupied by filler cells for DFM (design for manufacturing) purposes, so we assume the attacker

will leverage those "open" spaces to add attack-gates, as in relevant literature [3, 6, 8, 13, 17, 19].

A typical practice of design houses is to provide abstract information to foundries about the functionality of the chip under fabrication. A malicious entity can combine this information with public domain data about the design house (e.g., clients) and deduce the chip's instruction set architecture (ISA). By extracting the layout's gate-level netlist [15, 19], the attacker can search for suitable victim flip-flops or routed signals either by executing various test-benches in HDL simulators [19] or by reconstructing the high-level functionality of modules [12]. Taken as a whole, we show that an adversary experienced in IC design can successfully insert an IRT.

On the operational aspect, we assume the attacker is able to interface with the modified CPU after its on field deployment and deploy handling software similarly to the local access scenarios considered in Section II. Afterwards, the attacker can exercise the HT's capabilities, leveraging the reliability offered by the interrupt resilient triggering mechanism.

IV. Our Approach

The goal of a HT attack against a CPU is to reliably deliver the payload, while remaining unaffected by the normal microprocessor operation. For illustrative purposes, consider the scenario proposed by [19] where an attacker performs a privilege escalation attack to elevate the rights of a malicious process. For that, the trojan payload must flip the state of the CPU's privilege bits to elevate the execution rights of the handling software. In this scenario, the operational time of the handling process is mostly divided between the time spent to activate the capacitive based trigger and that spent on leveraging the HT's payload. In practice though, the operational period is not contiguous but rather a chain of time slices due to interleaved multitasking and kernel switching phenomena, as seen in Figure 1. Consequently, the noise induced in the attack's operational time is significant and unpredictable. What's more, a return to the the handling process does not guarantee the state of the privilege bits (payload) and the code location execution is resumed from. The outcome is that the CPU operational reality translates to uncertainty about the trigger signal state, which can jeopardize the trojan's stealthiness. *Thus, we view the trigger signal as the most time sensitive aspect of hardware trojan designs.*

A. Trigger Circuit Designs

To directly confront the aforementioned challenges, we introduce *interrupt-resilient trojans* or IRTs, that overcome the random context switching events through their unique CS-aware designs. In what follows, we characterize IRTs by their trigger mechanism and describe our design choice rationales.

1) Selectively-Ready Trigger (IRT-1): During context switching, the operating system is responsible for saving the state of the currently running process and restoring that of the next to be executed. We leverage the microarchitectural effects of this OS procedure to create IRT-1, a trigger mechanism implemented inside modules experiencing this non-stop cycle of state saves and restores. The requirement for an IRT-1 host module is that it should (*i*) participate in the state restore procedure upon the CPU's switch to the handling process, and (*ii*) be influenced through non-privileged instructions of the handling software for attack controllability.

CPU Modules	Instruction Influenced	CS Support
Adders [1]	✓	✗
Dividers	✓	✗
Multipliers	✓	✗
Program Counter	✓	✓
Floating Point Reg.	✓	✓
General Purpose Reg. [1]	✓	✓

TABLE I: Suitable CPU host modules for IRT-1 and IRT-2.

Host Module Requirements	IRT-1	IRT-2
CS Support	✓	✗
Inst. Influenced	✓	✓
Multi-bit Signals	✓[2]	✓[2]

TABLE II: Host module requirements for IRTs.

2) Always-Ready Trigger (IRT-2): As an alternative, we follow an "always-ready" strategy where the IRT-2 mechanism constantly delivers the trigger signal throughout the operational time of the attack. As for the IRT-2 host module, the only prerequisite is that it is comprised of hardware logic that can be readily influenced through non-privileged instructions. Hence, an IRT-2 can "infect" a larger set of microarchitecture modules, as evident in Table I. That said, this enhanced capability comes at the expense of the extra support logic (see Figure 2) that is required to maintain the trigger's ON state.

B. Selecting Signals for the Trigger Mechanism

In practice, settling on a way to implement a trigger's circuitry is non-trivial. To date, two approaches have been explored in the academic literature.

The first utilizes so-called *rare signals* (*e.g.,* signals with low toggling rate probability) for the creation of trigger circuits [6, 8, 11, 19]. The conjecture is that strongly biased signals decrease the probability of an inadvertent trojan triggering. To find these rare triggers, relevant research [6, 8] suggests signal profiling using representative benchmarks for CPU workloads [19]. Inevitably, the rare signal selection is biased by the benchmark used, and a shift in the workload can alter the rareness of signals that are relied upon.

The second approach [10, 13] involves monitoring general purpose hardware for the existence of specific *trigger values*. The rationale here is that accidental triggering can be reduced by increasing the number of bits [6] or the trigger sequences used. Conventional IRT host modules that can provide multi-bit signals for the trigger generation are shown in Table I. Importantly, this method does not require a priori knowledge of likely CPU workloads. Therefore, trojan activation is more streamlined, as long as the handling software can have immediate influence on the trigger host module. For our implementations, we choose the *triggering values* approach because it is more controllable and less onerous to implement during fabrication. A summary of all the IRT host module requirements is outlined in Table II for the reader's convenience.

[1]The examples we used for our evaluation in Section V.

[2]If triggering values are selected as the triggering approach.

C. Tackling Silicon Reality Constraints

Irrespective of the triggering approach, the fabrication stage imposes the same practical constraints on the attacker, namely:

i) The trigger and the payload host modules might be spatially separated on the layout, thereby requiring the time critical trigger signal to travel a significant distance. This in turn can lead to race conditions between the trigger signal and CPU events targeted by the payload.

ii) The *area or routing congestion* around host modules of interest might be high, leaving limited space for the placement and routing of trigger gates and nets.

iii) The signals of interest might belong in the circuit's *critical path*, with further utilization of them in the HT design, running the risk of generating timing violations.

With respect to the spatial locality challenge, the IRT-2 solution is immune due to its "always-ready" design. However, the IRT-1 solution is susceptible, as the trigger signal needs to travel to the payload each time the handling software resumes execution in the pipeline. To accommodate for that, the trigger should be treated as a *multi-cycle path*, meaning a data path sampled at a lower rate than that of the clock signal. Thus, multiple clock cycles are available for a valid value to show up at the end of such a path. Attackers can offer this option, by leveraging time expensive CPU events, to mask the time needed for the arrival of the trigger signal. Our proof of concept in Section V-A, uses the *page table walking* event to mask the arrival time of the IRT-1 trigger signal. In our evaluation, we attach the IRTs on a set of general purpose registers and the operands of an adder because they cover all of the requirements outlined in Table II.

V. Evaluation

To demonstrate the fact that IR-based trojans can operate in the presence of diverse context switching events while also overcoming the aforementioned silicon challenges, we couple our trigger designs with a payload that can undermine the integrity and availability of a CPU and attack the CVA6 [20] RISC-V microarchitecture. Specifically, the payload violates the separation mandated by the operating system between privileged and non-privileged areas of a CPU's memory. To that end, we target the exception generation mechanism within the memory management unit (MMU). The payload suppresses the exception signal generated for faulty store memory accesses that try to alter addresses whose privilege rights do not concur with the privilege state of the processor. To do so, we interfere with the User-mode bit (U-bit) of the page table entry (PTE) under access, and present a modified U-bit version to the exception handling module.

First, we perform an integrity attack that modifies arbitrary kernel space addresses selected by the attacker. These addresses might belong to different kernel modules running on the CPU and support a diverse set of software security mechanisms (*i.e.,* access control policies, packet filtering). To demonstrate the generalization of this attack, we perform an experiment in a controlled setting and target a custom made Linux kernel module (LKM). Our LKM allocates a certain set of addresses inside the kernel address space and the attacker uses the handling process to modify the contents of the allocated addresses with attacker-specific values. We implement this attack with both IRT-1 and IRT-2 trojans.

Second, we perform an attack that affects availability. In this experiment, we explicitly modify addresses containing kernel structures of type "*task_struct*". A "*task_struct*" element is a process descriptor containing information about a respective process and belongs to the kernel's task list. In this particular attack we overwrite the addresses following the "*init_task*" structure of the *init* task and cause a kernel panic.

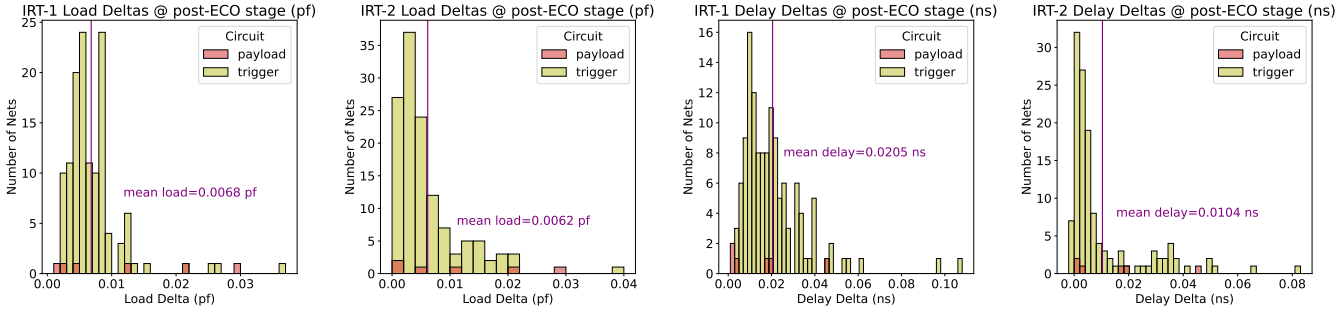
A. Resilience to Interrupts

In what follows, we evaluate our IRTs under both kernel context switching and multitasking scenarios. We run our experiments on a Genessys 2 FPGA board using a CVA6 design with integrated IRTs.

Kernel Context Switching: Abiding by the requirements of Table II, we implement IRT-1 inside CVA6's register file and attach it on the nets of two 64-bit general purpose registers (GPRs). Loading a specific 128-bit sequence in this register set instantiates the HT. A context switch out of the handling process temporarily removes the sequence from the register set until the handling software resumes execution. Overwriting of this sequence translates to deactivation of the trojan. Undoubtedly, this constant transition between ON and OFF states can lead to race conditions between the payload delivery and the targeted CPU events (*i.e.,* a faulty store memory access). Due to the fact that the trigger signal might travel a long distance to reach the payload (*e.g.,* because of spatial separation on the layout), the trigger must be treated as a multi-cycle path, as discussed in Section IV-C. To offer the extra clock cycles for the trigger propagation, we take advantage of the page table walking event. As the targeted LKM virtual addresses are not part of the handling software's address space, a page table walk is invoked with each overwrite to recover the physical addresses. This procedure requires multiple clock cycles to finish, providing sufficient time for the trigger signal to propagate and setup (re-enable) the payload. The payload can then suppress the exception signal generated by the MMU with every faulty store memory access in the kernel address space. Our experiments show that despite the multiple kernel context switching incidents, the IRT-1 trojan was able to successfully overwrite differing amounts of data in the kernel.

Process Context Switching: For the IRT-2 implementation to satisfy the requirements of Table II, we use the integer adder of CVA6's arithmetic logic unit (ALU) as the trojan host. We attach IRT-2 on the nets of the two 64-bit input operands. Starting the attack, we influence the adder's operands with a specific set of *activation* values, that force the FSM of Figure 2 to state S1 and enable the trigger. The trigger remains as such until the attack's end, where a specific set of *de-activation* values disable the trigger (FSM switches to S0).

Next, we consider the prevalence of interrupts (due to multitasking) that can occur while a HT attack is underway.



(a) Load (pf) overhead from IRT-1. (b) Load (pf) overhead from IRT-2. (c) Delay (ns) overhead from IRT-1. (d) Delay (ns) overhead from IRT-2.

Fig. 3: Load (pf) and delay (ns) overhead on original layout nets caused by IRT-1 and IRT-2.

	Hardware Trojan Characteristics							Physical Implementation Results				
Target	Trigger	# Trigger Bits	Trigger Host	Payload Host	# Comb. Cells	# Seq. Cells	# Conn. Nets [3]	Frequency (MHz) before	Density (%) before | after	Total Power (μW) before | after	Slack (ps) before | after	# Violations after
CVA6	IRT-1	128	Set of GPRs	MMU	50	4	55	600	75.32 | 76.537	241.31 | 241.44	0.00 | 0.00	0
	IRT-2	128	ALU Adder	MMU	64	6	70	600	75.32 | 76.539	241.31 | 244.70	0.00 | 0.00	0

TABLE III: Physical implementation results showing the impact of IR-based HTs on a CVA6 layout after insertion.

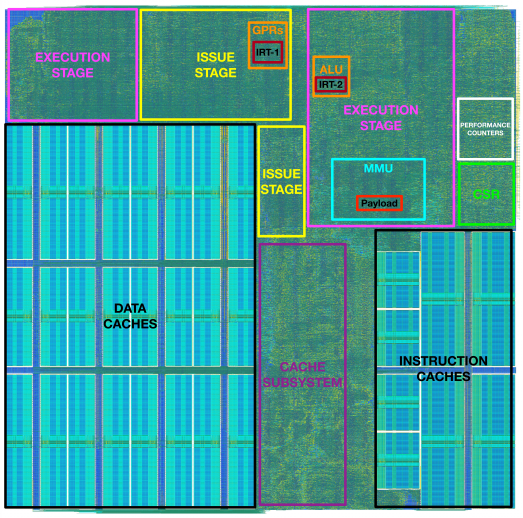


Fig. 4: Tape-out ready layout depicting both HT designs.

To emulate the multitasking operation, we interleave our HT handling software with the execution of other user level processes and the kernel. After, setting the IRT-2 generated trigger to an 'always-ready' mode (S1 state), we execute a program that randomly interleaves the handling software with the execution of a general computing performance benchmark (*e.g.,* Dhrystone). For an average of 11 KBs of overwritten kernel data, we witness a mean of 83 context switching events. Despite the heavy interrupt activity, the overwrite of arbitrary LKM addresses concludes successfully.

B. Reassessing Fabrication Stage Flexibility

To support our assertion that IRTs provide a pathway for fabrication stage attacks, we implement in a commercial 28nm process technology, the CVA6 tape-out ready layout (sign-off timings and no manufacturing violations) shown in Figure 4, to insert our trojans and measure their overall impact on the layout performance. The CVA6 layout represents the GDSII file received at the foundry from a trusted design house and is both high speed (clock frequency of 600MHz) and high density (>75% core utilization).

Our trigger circuit implementations use the strategy suggested by Su et al. [16], wherein they analyze different logic gate connection patterns and record those with very low transition probability on their output. We use as cores of IRT-1 and IRT-2 circuits, the recommended connection patterns $\langle$`AND` $\rightarrow$ `NAND`$\rangle$, $\langle$`NAND` $\rightarrow$ `NOR`$\rangle$. This selection leads to a very low (inadvertent) activation probability and thus, enhances the HTs' stealthiness. We couple IRT-1 and IRT-2 with the payload of Section V and insert them in two separate CVA6 layouts.

Our threat model considers an adversary that gains access on the finalized layout at the foundry and performs an IRT insertion in the open areas of the layout. To emulate *the experience and expertise of a skilled adversary attempting a manual IRT insertion* we use the pre-mask ECO flow that P&R tools provide, since ECOs *(i)* keep the original layout intact by swapping only filler cells with attack gates and *(ii)* deteriorate marginally the layout timings during routing of attack nets.

An important concern for the adversary is the coupling capacitance inserted from the HT wiring. The use of ECOs naturally minimizes any such impact. For a manual insertion, we argue that the small number of IRT gates and wires in Table III helps keeping the inserted coupling capacitance to minimum. Besides, coupling capacitance is a well studied topic in the microelectronics community [5], with reduction techniques that attackers can utilize (e.g., wire sizing, wire spacing, adding repeaters, metal layer jogging). Thus, our use of ECOs resembles a realistic manual HT insertion from a knowledgeable adversary. After the HT insertion, we perform sign-off static timing analysis and DRC checks to verify that

[3]This is the number of the interconnects between the HT gates.

the initial fabrication standards are unaffected. The impact of IR-based HTs on the CVA6 layout is summarized in Table III. The discrepancy in the number of attack cells between the two IRTs is attributed to the extra support logic of IRT-2 (Figure 2) and the repeater cells of the further away placed IRT-1 (Figure 4). Our results are comparable to those of Hepp et al. [8]. In particular, despite the austere critical path of CVA6's layout (0ns vs 15ns slack for PULPino in [8]), IRTs do not introduce any timing violations (unlike PULPino's case).

To further assess the feasibility of fabrication stage IRTs, we compute the capacitive load and time delay overheads (Figures 3-a, 3-b and Figures 3-c, 3-d) on the original nets that IRTs' attach to (GPR's and adder operands). The average load added on this layout and technology process is 6.8fF for IRT-1 and 6.2fF for IRT-2, which yields an average extra delay on the nets of 20ps and 10ps respectively. This observation suggests that IRTs can target very high frequency CPU layouts (*e.g.,* a 20ps overhead is only 3.4% of the original 1.7GHz clock cycle in [20]). Overall, our measurements show that IRT designs are minimalistic enough for fabrication stage attacks.

Lastly, we note that the CVA6 microarchitecture [20] was designed with a flip-flop-based register file. This design choice allows using the nets coming out of the flip-flops for the implementation of IRT-1. A different practice involves the use of SRAM-based register files, wherein IRT-1's structure is unsuitable. However this limitation does not detract from the generality of IRT attacks, in the same way that the existence of a single-bit register for privilege escalation in the OR1200 microarchitecture [2] does not detract from the generality of the A2 trojan attacks [19]. Moreover, attackers can target other general purpose modules of Table I.

VI. Conclusions

In this work, we show how context switching events can hinder the successful execution of HT attacks against CPUs. To counter that threat, we introduce the concept of Interrupt Resilient Trojans (IRTs) that can withstand non-deterministic context switching events while reliably delivering payloads. By showing that IRTs can be efficiently integrated inside tape-out ready layouts, we demonstrate that fabrication stage attacks can indeed be quite flexible. To promote research in this area, we make our implementation publicly availabile [1].

References

[1] Interrupt Resilient Hardware Trojans. https://github.com/0ena/riscv-hw-trojans.

[2] Openrisc 1200. https://github.com/openrisc/or1200.

[3] A. De, M. N. I. Khan, K. Nagarajan, and S. Ghosh. Hartbleed: Using hardware trojans for data leakage exploits. *IEEE Trans. on Very Large Scale Integ. Systems*, 2020.

[4] K. Dharsee and J. Criswell. Jinn: Hijacking safe programs with trojans. In *USENIX Security Symposium*, 2023.

[5] M. Elgamel and M. Bayoumi. Interconnect noise analysis and optimization in deep submicron technology. *IEEE Circuits and Systems Magazine*, 2003.

[6] V. Gohil, H. Guo, S. Patnaik, and J. Rajendran. Attrition: Attacking static hardware trojan detection techniques using reinforcement learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[7] A. Hepp and G. Sigl. Tapeout of a RISC-V crypto chip with hardware trojans: a case-study on trojan design and pre-silicon detectability. In *Computing Frontiers Conference*, 2021.

[8] A. Hepp, T. Perez, S. Pagliarini, and G. Sigl. A pragmatic methodology for blind hardware trojan insertion in finalized layouts. In *IEEE/ACM International Conference on Computer-Aided Design*, 2022.

[9] N. Jacob, D. Merli, J. Heyszl, and G. Sigl. Hardware trojans: current challenges and approaches. *IET Comput. Digit. Tech.*, 2014.

[10] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[11] C. Kison, O. M. Awad, M. Fyrbiak, and C. Paar. Security implications of intentional capacitive crosstalk. *IEEE Trans. Inf. Forensics Secur.*, 2019.

[12] T. Meade, S. Zhang, and Y. Jin. Netlist reverse engineering for high-level functionality reconstruction. In *Asia and South Pacific Design Automation Conference*, 2016.

[13] A. Moschos, K. Valakuzhy, and A. D. Keromytis. On the feasibility of remotely triggered automotive hardware trojans. In *Int. Conference on Electrical, Computer, Communications and Mechatronics Engineering*, 2022.

[14] S. Parvin, M. Goli, F. S. Torres, and R. Drechsler. Trojan-D2: Post-Layout Design and Detection of Stealthy Hardware Trojans - A RISC-V Case Study. In *Asia and South Pacific Design Automation Conference*, 2023.

[15] R. S. Rajarathnam, Y. Lin, Y. Jin, and D. Z. Pan. Regds: A reverse engineering framework from gdsii to gate-level netlist. In *IEEE International Symposium on Hardware Oriented Security and Trust*, 2020.

[16] Y. Su, H. Shen, R. Lu, and Y. Ye. A stealthy hardware trojan design and corresponding detection method. In *IEEE Int. Symposium on Circuits and Systems*, 2021.

[17] N. G. Tsoutsos and M. Maniatakos. Fabrication attacks: Zero-overhead malicious modifications enabling modern microprocessor privilege escalation. *IEEE Trans. Emerg. Top. Comput.*, 2014.

[18] M. Xue, C. Gu, W. Liu, S. Yu, and M. O'Neill. Ten years of hardware trojans: a survey from the attacker's perspective. *IET Comput. Digit. Tech.*, 2020.

[19] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester. A2: Analog malicious hardware. In *IEEE Symposium on Security and Privacy*, 2016.

[20] F. Zaruba and L. Benini. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Trans. on Very Large Scale Integ. Systems*, 2019.