To Appear in 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS), November 2022

ATTRITION: Attacking Static Hardware Trojan Detection Techniques Using Reinforcement Learning

Vasudev Gohil, Hao Guo, Satwik Patnaik, and Jeyavijayan (JV) Rajendran
Electrical & Computer Engineering, Texas A&M University, College Station, Texas, USA
{gohil.vasudev,guohao2019,satwik.patnaik,jv.rajendran}@tamu.edu

ABSTRACT

Stealthy hardware Trojans (HTs) inserted during the fabrication of integrated circuits can bypass the security of critical infrastructures. Although researchers have proposed many techniques to detect HTs, several critical limitations exist, including: (i) a low success rate of HT detection, (ii) high algorithmic complexity, and (iii) a large number of test patterns. Furthermore, as we show in this work the most pertinent drawback of prior (including state-of-the-art) detection techniques stems from an incorrect evaluation methodology, *i.e.,* they assume that an adversary inserts HTs randomly. Such inappropriate adversarial assumptions enable detection techniques to claim high HT detection accuracy, leading to a *"false sense of security." To the best of our knowledge, despite more than a decade of research on detecting HTs inserted during fabrication, there have been no concerted efforts to perform a systematic evaluation of HT detection techniques.*

In this paper, we play the role of a realistic adversary and question the efficacy of HT detection techniques by developing an automated, scalable, and practical attack framework, ATTRITION, using reinforcement learning (RL). ATTRITION evades eight detection techniques (published in premier security venues, well-cited in academia, etc.) across two HT detection categories, showcasing its agnostic behavior. ATTRITION achieves average attack success rates of 47× and 211× compared to randomly inserted HTs against state-of-the-art logic testing and side channel techniques. To demonstrate ATTRITION's ability in evading detection techniques, we evaluate different designs ranging from the widely-used academic suites (ISCAS-85, ISCAS-89) to larger designs such as the open-source `MIPS` and `mor1kx` processors to `AES` and a `GPS` module. Additionally, we showcase the impact of ATTRITION generated HTs through two case studies (privilege escalation and kill switch) on `mor1kx` processor. We envision that our work, along with our released HT benchmarks and models (post peer-review), fosters the development of better HT detection techniques.

CCS CONCEPTS

• **Security and privacy** → **Malicious design modifications**;

KEYWORDS

Hardware Trojans, Reinforcement Learning, Attacks

1 INTRODUCTION

1.1 Globalized IC Supply Chain and Threats

Integrated circuits (ICs) are the backbone of modern computing systems. Enabling high-performance and low-power ICs requires access to smaller and faster transistors (building blocks of ICs). The push for continual miniaturization of transistors necessitates reliance on state-of-the-art fabrication facilities, also known as foundries. However, commissioning a state-of-the-art foundry incurs astronomical costs [1, 2]. For instance, TSMC, the world's largest contract IC manufacturer, has allocated $40 billion to increase chip production for the state-of-the-art 2nm technology node [2]. To reduce the design cost and ameliorate marketing constraints, leading IC design companies like Apple, Qualcomm, and NVIDIA operate in a fabless model [3] and outsource IC manufacturing to off-shore, third-party foundries, which could be potentially *untrustworthy* (refer to Appendix A.1 for more details). The U.S. *Department of Defense (DoD)* 2022 action plan for securing defense-critical supply chains indicates that 88% of microelectronic fabrication is performed overseas, representing a substantive security threat [4]. Such a distributed supply chain has given rise to numerous security concerns ranging from IP piracy [5–7] to the insertion of malicious logic known as hardware Trojans (HTs) [8–13].

1.2 Disruptive Impact of Hardware Trojans

HTs, once inserted during fabrication, cannot be removed. Unlike buggy software, where a software patch can help mitigate the ill effects, the damage incurred by a stealthy HT has far-reaching consequences [10, 12, 14, 15]. For instance, an HT enables privilege escalation [12, 16] or an HT implanted in critical infrastructure (e.g., radar systems) leads to national security threats [17]. To highlight the practicality and importance, researchers have discovered and demonstrated HTs in silicon [12, 14, 15]. For example, [14] found a "backdoor" in a military-grade chip, [15] crafted HTs that compromised cryptographically-secure random number generators used in Intel's *Ivy Bridge* processors, and [12] performed privilege escalation using a capacitor-based HT on fabricated chips. These examples showcase the pernicious effects of HTs in bypassing the security of a system. Apart from academic endeavors, the *Defense Advanced Research Projects Agency* (DARPA), a research and development agency of the U.S. DoD, is accelerating HT research through programs like *Safeguards against Hidden Effects and Anomalous Trojans in Hardware* (SHEATH) [18] and *Automatic Implementation of Secure Silicon* (AISS) [19], the latter in collaboration with Synopsys, a leading electronic design automation company.

1.3 Hardware Trojan Detection Techniques

Researchers attempt to detect HTs inserted during fabrication using (i) logic testing [20–23] and (ii) side channel analysis [24, 25, 27–33]. Both approaches apply input patterns to the underlying chip, and the only difference is the modality they monitor. In a logic testing-based approach, the defender monitors the output responses from the chip to detect deviations from the expected outputs using an HT-free (golden) chip [20–23]. In a side channel-based approach, the

Table 1: HT detection techniques against ATTRITION. ✓ indicates that ATTRITION evades detection.

Type	Logic testing				Side channel			
Detection Technique	MERO [20]	GA+SAT [21]	TARMAC [22]	TGRL [23]	MERS [24]	MERS-h [24]	MERS-s [24]	MaxSense [25]
Venue	CHES'09	CHES'15	TCAD'21	ASP-DAC'21	CCS'16	CCS'16	CCS'16	TODAES'21
Algorithm	Random bit-flipping	Genetic algorithm	Maximal clique enumeration	Reinforcement learning	Random bit-flipping	Reordering using Hamming distance	Reordering using simulation	Genetic algorithm
Largest Design (# gates)	s35932† (6,500)	s38417† (22,179)	MIPS (≈25,000)	s35932† (12,204)	s35932† (6,500)	s35932† (6,500)	s35932† (6,500)	MIPS (≈25,000)
Evaluation Methodology	Randomly-inserted HTs	Randomly-inserted HTs	Randomly-inserted HTs	Randomly-inserted HTs	Randomly-inserted HTs	Randomly-inserted HTs	Randomly-inserted HTs	Randomly-inserted HTs
Claimed defense Efficacy	93.5%	79.4%	93.1%	96.1%	'... MERS is effective for any Trojan forms/sizes ..." [24]			96.6%
ATTRITION (This Work)	✓	✓	✓	✓	✓	✓	✓	✓

†These designs are from ISCAS-89 benchmark suite, widely-used by the hardware security community [5, 7, 26].

defender monitors the physical characteristics (power consumed, path delay, electromagnetic emissions) of the chip to detect deviations from expected behavior [24, 25, 27–33].

Limitations of Existing Techniques. The aforementioned HT detection techniques suffer from key limitations, including: (i) low success rate for detecting HTs [20, 21, 24], (ii) high algorithmic complexity [21, 22, 24, 25], and (iii) a large number of input patterns [20–24], leading to increased test time and delayed deployment. However, the most pertinent drawback arises from a lack of proper security evaluation of these detection techniques, resulting from inappropriate/incorrect adversarial assumptions (§2.3). The ramifications of an inappropriate security evaluation lead to a "*false sense of security*." To the best of our knowledge, despite more than a decade of research on detecting HTs (implanted during fabrication), there have been no concerted efforts to perform a systematic and automated evaluation of HT detection techniques that can scale well to industrial-scale designs.

1.4 Our Goals and Contributions

In this work, we perform a systematic evaluation of HT detection techniques that aim to detect HTs during the fabrication of ICs. We assume the role of a motivated and realistic adversary that inserts HTs to evade state-of-the-art HT detection techniques. However, detection techniques involve various algorithms ranging from (i) random bit-flipping [20, 24], (ii) genetic algorithms [21, 25], (iii) graph-theoretic algorithms [22], to (iv) reinforcement learning [23]. *Therefore, we need a radical approach to systematically evaluate HT detection techniques by inserting stealthy HTs using an automated attack framework.*

However, several hurdles exist in designing an automated attack framework that inserts stealthy HTs during fabrication. First, the underlying design is a sea of gates and nets; it is computationally challenging to examine each net individually to create an HT. Second, the input patterns generated by the HT detection techniques are not deterministic. Third, an adversary does not have apriori information regarding the locations tested by the defender. This moving target (*i.e.,* the stochasticity of locations checked by the defender, non-deterministic input patterns, large design space exploration) makes it challenging for an adversary to evade detection techniques.

We address the aforementioned hurdles and develop a scalable attack framework, ATTRITION, using reinforcement learning (RL). RL has shown great promise in navigating unknown and uncertain problem spaces and finding optimal or near-optimal solutions, as in the case of fuzzing [34, 35], Internet of Things security [36, 37], and cyber security [38, 39]. Hence, we formulate the task of inserting HTs, which evade the detection techniques under the uncertainty of the input patterns, as an RL problem (§4). However, we must overcome several challenges to realize an automated, practical, and scalable RL agent: ① dependence on detection techniques, *i.e.,* reliance on input patterns from detection techniques, ② expensive reward computations, *i.e.,* evaluation of HTs generated during training, ③ inefficiencies of the agent's choices, *i.e.,* inhibiting the agent from generating HTs that are unsuitable, ④ lack of scalability, *i.e.,* insert HTs in large designs like AES, GPS, and mor1kx processor), and ⑤ lack of variety of HTs, *i.e.,* generate a large corpus of HTs for an adversary to choose from.

We overcome the challenges of ① reliance on input patterns and ② expensive reward computations by characterizing the design before training. Characterization helps us compute the rewards quickly (up to 16× faster than the naïve approach) (§4.3). To reduce the ③ inefficient choices made by the agent, we trim the actions available to the agent at different time steps depending on the present state of the agent (§4.4). Finally, we overcome challenges about ④ scalability and ⑤ limited variety of HTs by carefully pruning the search space for the agent (§4.5).

By solving these challenges, we develop an automated and scalable RL-based framework, ATTRITION, that performs a systematic evaluation of HT detection techniques by inserting stealthy HTs. ATTRITION is agnostic to the choice of detection techniques considered in this work and scalable to practical designs, such as AES, GPS, and mor1kx processor. ATTRITION generated HTs evade eight HT detection techniques from logic testing (§5.2) and side channel-based detection approaches (§5.3), and we showcase two case studies to demonstrate cross-layer, system-level attacks on mor1kx processor (§5.6). The **contributions** of our work are as follows.

- We develop an automated, scalable, and practical RL-enabled HT insertion framework, ATTRITION , that successfully evades several HT detection techniques, including the state-of-the-art. To the best of our knowledge, our work is the first to use RL for developing a successful attack in supply chain security (§4).
- We demonstrate the generalization power of ATTRITION by evading eight HT detection techniques from logic testing and side

channel categories (Table 1). These techniques have been published in premier security venues, have been widely regarded in the industry, and cited in academia, and/or are state-of-the-art. ATTRITION achieves average attack success rates of 47× and 211× compared to randomly inserted HTs against state-of-the-art logic testing and side channel techniques (§5.2 and §5.3).
- We showcase the efficacy of ATTRITION generated HTs on designs ranging from the widely used ISCAS-85 and ISCAS-89 benchmark suites to open-source `MIPS` and `mor1kx` processors, `AES`, and `GPS` (upto $\approx 200,000$ gates; §5).
- We demonstrate how an adversary can repurpose ATTRITION to design HTs that not only evade detection, but also cause practical, cross-layer, real-world attacks through two case studies (privilege escalation and kill switch) on `mor1kx` processor (§5.6).
- To foster further research in the area of HT detection and HT insertion techniques, we will open-source our codes and HT-infested designs. Our developed HTs can be used by researchers to evaluate the efficacy of their detection techniques.

2 BACKGROUND AND PRELIMINARIES

We first provide an overview of hardware Trojans (HTs) and explain different HT detection techniques and their evaluation methodologies, followed by an introduction to reinforcement learning (RL).

2.1 Hardware Trojans (HTs)

HTs are malicious logic inserted by adversaries to achieve a disruptive impact on ICs [8, 10, 16, 40]. HTs can (i) operate as a hardware backdoor that leaks sensitive information such as cryptographic keys [41], (ii) cause denial-of-service during regular operation [42], and/or (iii) effect a deviation in the functionality of the design [12]. Typically HTs can be inserted anywhere in the IC supply chain ranging from the register-transfer level (adversary is the third-party intellectual property provider) to the chip layouts (adversary in the foundry). A taxonomy of HTs can be found in the Appendix A.2. An HT consists of two components: **trigger** and **payload**. The trigger is the activation mechanism of an HT, and the trigger is activated by **rare nets** (*i.e.,* nets whose logic values are strongly biased). In particular, the trigger is activated when the rare nets assume their rare values. These rare nets have an **activity probability** (*i.e.,* the probability of a net being a 1 or 0) below a certain **rareness threshold**. Once the trigger is excited, the payload gets activated, causing a malicious effect. A **stealthy HT** must be (i) malicious (jeopardize the chip's functionality or leak sensitive information or degrade the performance of the device) and (ii) undetectable (would not be detected by any prior and state-of-the-art HT detection techniques).

2.2 Prior Work on Hardware Trojan Detection

When the foundry is untrusted, HT detection techniques are classified into two broad categories: logic testing and side channel analysis. Logic testing-based techniques detect HTs by applying test patterns to the HT-infested design to activate the trigger [20–23]. On the other hand, side channel-based detection techniques detect HTs by monitoring the deviations of the side-channel measurements (power consumption, path delay, electromagnetic emissions) of an HT-infested design from the expected measurements of a golden, *i.e.,* HT-free, design [24, 25, 27–33].

Although there are many noteworthy HT detection techniques for both categories, we choose eight techniques for our evaluation, as explained next. Our selection spans from the earliest technique with industrial adoption, namely MERO [20], to the latest one that uses an RL algorithm as a detection tool, namely TGRL [23]. We also select techniques that have a high impact (measured through citations) from a diverse set of conferences/journals: (i) MERS [24] from ACM CCS, (ii) TARMAC [22], an industry-adopted technique [43], from IEEE TCAD, a top computer-aided system design transactions, (iii) GA+SAT [21] from CHES, a top hardware security conference, and (iv) MaxSense [25], the state-of-the-art technique for side channel-based HT detection (62× better than MERS [24]). These techniques claim the following properties: (i) scalability, (ii) efficiency, (iii) accuracy in detecting stealthy HTs, and (iv) ease of integration with IC design tools and flow. In addition to MERS and MaxSense, there are a few other high-impact works on detecting HTs using side-channel analysis, such as [29, 31, 44, 45]. However, since MERS and MaxSense magnify the impact of all of these techniques, we chose MERS [24] and MaxSense [25].

Logic testing-based Techniques. MERO generates test patterns that activate each rare net K times [20]. The hypothesis is that if all the rare nets are activated K times, the resultant test patterns are likely to activate unknown triggers. **GA+SAT** generates test patterns by utilizing a genetic algorithm and Boolean satisfiability (SAT) [21]. **TARMAC** generates test patterns using clique enumeration [22]. **TGRL** uses RL to generate a set of test patterns to maximize the likelihood of activating HTs [23].

Researchers evaluate the efficacy of logic testing-based detection techniques using **HT activation rate** [20–23], defined as the percentage of HTs activated by the test patterns. Mathematically, the HT activation rate is $\left(\frac{\text{Number of HTs activated}}{\text{Total number of HTs inserted}}\right) \times 100\%$.

Side channel-based Techniques. MERS extends the idea of MERO to side-channel metrics by generating test patterns that cause rare nets to switch from their non-rare to rare values K times [24]. Doing so increases the likelihood of activating an HT, which in turn increases the deviation of the measured side-channel from expected (*i.e.,* golden) values. **MERS-h** reorders test patterns generated by MERS to simultaneously maximize the activity in rare nets and minimize the activity in non-rare nets. Unlike MERS-h, **MERS-s** relies on the actual behavior of the golden circuit (obtained through functional simulations) to measure the activities in the rare and non-rare nets. **MaxSense** exploits input affinity to generate test patterns that maximize switching in the malicious logic while minimizing switching in the rest of the circuit [25].

Researchers evaluate the efficacy of side channel-based detection techniques using **side-channel sensitivity** [24, 25], which measures the amount of switching caused in an HT-infested design relative to the switching caused in an HT-free design. Let G be the golden design (*i.e.,* HT-free), HT be the HT-infested design, and (u_i, v_i) denote the i^{th} pair of consecutive test patterns, then, the side-channel sensitivity of the HT-infested design HT is

$$sensitivity_{HT} = \max_{(u_i,v_i)} \left(\frac{|switching^{HT}_{(u_i,v_i)} - switching^{G}_{(u_i,v_i)}|}{switching^{G}_{(u_i,v_i)}} \right)$$

We use these metrics (**HT activation rate** and **side-channel sensitivity**) to showcase the efficacy of ATTRITION.

2.3 Evaluation Methodology of Detection Techniques

The aforementioned detection techniques span over a decade—from 2009 to 2021. Collectively, they cover several algorithms, ranging from random bit-flipping [20], genetic algorithm [21, 25], graph-theoretic algorithm [22], reinforcement learning-based algorithm [23], to side channel-based detection [24, 25]. However, all these techniques lack proper evaluation in two aspects.

Firstly, all aforementioned detection techniques assume that an adversary randomly samples HTs from the rare nets. As a result, these detection techniques claim very high efficacy (> 90% HT activation rate). Similarly, benchmark suites of HT-infested designs (e.g., TrustHub [46, 47] and [48]) consist of randomly inserted HTs. However, this assumption does not reflect the real-world scenario since an adversary in a foundry is not constrained to insert HTs randomly, as evidenced in [12, 14–16]. On the contrary, an adversary inserts HTs such that the likelihood of detection is minimal. Our results demonstrate that the HTs generated by ATTRITION cause a drastic reduction in the efficacy of logic testing-based techniques (Table 4) and side channel-based techniques (Table 5).

Secondly, all aforementioned detection techniques consider designs that contain just a few thousand gates. Such designs are not practical since modern design intellectual property cores contain at least a few hundred thousand gates (e.g., `AES`, `GPS`, `mor1kx` processor). Furthermore, detecting HTs in designs containing a few thousand gates is relatively easier since there are limited places where an HT can be inserted. Thus, the evaluation methodology used by the state-of-the-art HT detection techniques is inappropriate, and it gives a *"false sense of security."* We use RL as a litmus test to demonstrate these issues and obtain the actual efficacy of HT detection techniques.

2.4 Reinforcement Learning (RL)

RL is a machine learning technique where an agent learns how to act in an (unknown) environment through actions and receives feedback through rewards. Unlike supervised/unsupervised learning, which requires labeled/unlabeled data; RL does not require pre-defined data. RL is used to solve problems involving an optimal sequence of decisions by modeling the underlying problem as a Markov decision process [49, 50]. While many fields such as software fuzzing [34, 35], Internet-of-Things security [36, 37], and cyber security[38, 39] have reaped the benefits of using RL, hardware security is still in its infancy to reap the powers of RL.

3 THREAT MODEL

Before diving into the specifics regarding ATTRITION, we outline the location, capabilities, and goal of an adversary.

Adversary Location. We assume the adversary is present in the untrustworthy foundry. Our threat model selection is motivated by the fact that most IC design companies outsource fabrication to overseas foundries [3] (§1). In fact, the U.S. *Department of Defense* 2022 action plan for securing defense-critical supply chains points out that 88% of microelectronic fabrication is performed overseas, representing a substantive security threat [4].

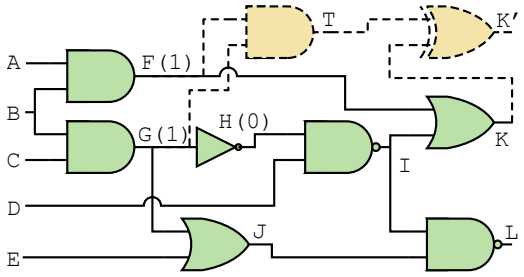

Figure 1: The trigger `T` (constructed from `F` and `G`), when activated, triggers the payload to flip the value of `K`.

Adversarial Capabilities. We outline the capabilities of an adversary consistent with state-of-the-art research in hardware Trojans (HTs) [20–25].

1. An adversary obtains the gate-level design by reverse-engineering the Graphics Database System II (GDSII).[1] The adversary has access to and know-how regarding state-of-the-art reverse engineering equipment [51–53].
2. An adversary can construct the trigger using only rare nets. To that end, they can compute the list of rare nets by performing functional simulations using any tool (academic or commercial).
3. An adversary has resources (placement sites for trigger and payload and routing resources for connecting wires) available in the GDSII to insert the trigger and payload.
4. An adversary does not know the input patterns used by the defender for post-fabrication testing. We assume that an adversary knows the type of HT detection technique(s) used by the defender.

Adversarial Goal. The goal of the adversary is to cause a disruptive impact (e.g., leak secret information like cryptographic keys [41], perform privilege escalation [12, 16]) on the ICs by inserting HTs while evading all detection techniques employed by the defender. Our work focuses on additive HTs since all considered detection techniques assume additive HTs [20–25, 40].

4 ATTRITION: ATTACKING HARDWARE TROJAN DETECTION TECHNIQUES

We initially explain why reinforcement learning (RL) is suited for our problem definition and our preliminary formulation; however, our preliminary formulation has several limitations. Subsequently, we explain the challenges and the steps we took to overcome them, and finally we outline the architecture for ATTRITION.

4.1 Why Reinforcement Learning?

Inserting stealthy hardware Trojans (HTs) (*i.e.,* evading HT detection techniques) necessitates the judicious selection of trigger nets (rare nets that constitute the trigger). A concerted selection strategy of trigger nets has two features, as explained next.

Sequential Decision Making. If an adversary wants to construct the trigger using α rare nets, a naïve approach would be to select all α nets simultaneously. However, since nets are part of the design, their logic values are dependent on each other. Thus, naïvely selecting α rare nets would be sub-optimal. For instance, consider

[1] GDSII is a database file used for exchanging IC layout information.

the design in Figure 1, where there are three rare nets (F, G, H with rare values 1, 1, and 0, respectively). However, the value of H depends on the value of G. In fact, whenever G takes its rare value, H also takes its rare value. Thus, selecting both these nets would be sub-optimal because any test pattern that activates G will also activate H, leading to easy detection. Alternatively, depending on the design's structure (*i.e.,* connectivity among logic gates), selecting one particular rare net can lead to a non-selection of other rare nets to form a valid HT (an HT that can be activated). Hence, a better approach would be to choose the rare nets sequentially, *i.e.,* select one rare net, understand its impact on the other rare nets in the design, and then select the next rare net. In other words, a sequence of optimal decisions is required to construct an HT that evades detection techniques.

Secondly, there is an **uncertainty in the test patterns** generated by the detection techniques, *i.e.,* the potential triggers they check for are *unknown to an adversary.* For instance, in a design with 1000 rare nets, constructing an HT with four rare nets can be done in ${}^{1000}C_4 \approx 4.1 \times 10^{10}$ ways. However, an adversary does not know the combinations checked by the defender. Hence, a concerted strategy should construct HTs (*i.e.,* select trigger nets) under the uncertainty of the locations checked by the detection techniques.

Problems with these two features (**sequential decision making** and **uncertainty in the environment**) are the kind of tasks that RL algorithms solve well—making an optimal sequence of decisions under uncertainty [49, 50]. Hence, we formulate the HT generation problem as an RL problem.

4.2 Preliminary Formulation

We now explain our preliminary formulation. We assume that an adversary constructs a trigger using T_{wid} rare nets,[2] throughout this section. Recall that, to evade existing detection techniques; we need to construct the trigger out of those combinations of rare nets that are not simultaneously activated by test patterns from existing detection techniques (§4.1). Note that if an adversary knew the *exact test patterns that a defender used to detect HTs*, the solution is straightforward—generate a trigger from rare nets not activated by the test patterns.

Since an adversary knows the HT detection technique(s) used by the defender (§3), they generate a set of test patterns, which will be different from the patterns used by the defender. Given these test patterns, an adversary aims to generate triggers (and hence HTs) that are unlikely to be detected by the defender. We construct the HT triggers by mapping the HT generation problem as an RL problem, as described below.

- **States** $\mathcal{S}$ is the set of all subsets of rare nets. An individual state s_t represents the set of rare nets chosen by the agent at time t.
- **Actions** $\mathcal{A}$ is the set of all rare nets. An individual action a_t is the rare net chosen by the agent at time t.
- **State transition** $\mathcal{P}(s_{t+1}|a_t, s_t)$ is the probability that action a_t in state s_t leads to the state s_{t+1}. If the chosen rare net (*i.e.,* the action) is **compatible**[3] with the current set of rare nets (*i.e.,* the current state), we add the chosen rare net to the set of compatible rare nets (*i.e.,* the next state). Otherwise, the next state remains the same as the current state. Thus,

$$s_{t+1} = \begin{cases} \{a_t\} \cup s_t, & \text{if } a_t \text{ is compatible with } s_t \\ s_t, & \text{otherwise} \end{cases}$$

- **Reward function** $\mathcal{R}(s_t, a_t)$**.** Denote $\mathcal{I}_C : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}$. Given any state action pair (s, a), $\mathcal{I}_C(s, a)$ indicates whether a is compatible with s or not. Denote $\mathcal{I}_A^{TP} : \mathcal{S} \rightarrow \{0, 1\}$. Given a state s, $\mathcal{I}_A^{TP}(s)$ indicates if s is activated by some test pattern in TP or not. Then, the reward function is defined as

$$\mathcal{R}(s_t, a_t) = \begin{cases} 0, & \text{if } (|s_{t+1}| \neq T_{wid}) \ \wedge \ (\mathcal{I}_C(s_t, a_t) = 1) \\ \rho_1, & \text{if } (|s_{t+1}| \neq T_{wid}) \ \wedge \ (\mathcal{I}_C(s_t, a_t) = 0) \\ 0, & \text{if } (|s_{t+1}| = T_{wid}) \ \wedge \ (\mathcal{I}_A^{TP}(s_{t+1}) = 1) \\ \rho_2, & \text{if } (|s_{t+1}| = T_{wid}) \ \wedge \ (\mathcal{I}_A^{TP}(s_{t+1}) = 0) \end{cases}$$

  Note that $\rho_1 < 0$, *i.e.,* a penalty, and $\rho_2 > 0$. Also, TP denotes a set of test patterns from a detection technique (e.g., TGRL [23]). We design the reward function so that the agent tries to construct triggers that are not activated by the generated test patterns.
- **Discount factor** γ $(0 \leq \gamma \leq 1)$ indicates the importance of future rewards relative to the current reward.

In the training process, we initialize the agent with a random rare net, and it chooses other rare nets at each step. At the end of the episode (*i.e.,* after $T_{wid} - 1$ steps), the rare nets chosen by the agent (so far) reflect the generated HT.

Challenges. Although this preliminary formulation achieves a high success rate in evading HT detection techniques (e.g., on average, our RL-generated HTs are 49.87× more stealthy against TARMAC than random HTs for c6288, c7552, s13207, and s15850,[4] it faces the following challenges: ① it relies on test patterns from detection techniques, ② simulations for reward computation are expensive, leading to large training time for medium-sized designs (e.g., agent does not generate stealthy HTs for MIPS with $\approx$ 25,000 gates even after 12 hours of training), ③ it is inefficient in choosing rare nets because the agent chooses some rare nets that are incompatible, ④ it is not scalable to large designs like processor cores or crypto-engines (e.g., mor1kx and AES), and ⑤ it generates only 67 stealthy HTs for mor1kx,[5] thus limiting the options for an adversary. Next, we explain these challenges and how we overcome them.

4.3 Offline Characterization

Challenge ①. A fundamental limitation of our preliminary formulation is that it relies on test patterns from a detection technique like TGRL [23] or TARMAC [22] for training. Ideally, the RL agent should be agnostic to the test patterns, thereby agnostic to the HT detection technique. If the agent relies on test patterns and researchers develop a better detection technique, the agent would need to be retrained with test patterns from this new technique.

[2] The number of rare nets used to construct the trigger is defined as **trigger width**.

[3] A set of rare nets is compatible if an input pattern exists that can activate all the rare nets in the set simultaneously. Alternatively, we say that a rare net, r_i, is compatible with a given set of rare nets, R, if an input pattern exists that can activate r_i and all nets in R simultaneously.

[4] Widely used designs by hardware security community [5, 7, 26].

[5] An adversary would prefer to have a corpus of stealthy HTs since some HTs might be infeasible for insertion due to resource constraints (lack of placement sites for trigger and/or lack of routing resources) or cause noticeable changes in side channels, thereby facilitating detection.

Table 2: Comparison of training rates for online and offline reward calculation methods for MIPS.

Reward calculation method	Training rate	
	steps/min	episodes/min
Online reward	≈ 13.88	≈ 4.63
Offline reward	≈ 234.82	≈ 78.23
Speedup	**16.91×**	**16.89×**

However, retraining is time-consuming since one needs to do it for every new technique. Hence, we aim to develop an agent that is agnostic to test patterns from detection techniques so that the agent can be used against other techniques in the future.

Challenge ②. Another reason for devising the agent to be agnostic to test patterns is time complexity for reward computation. In our preliminary formulation, computing the reward requires us to check if the final state of the agent, *i.e.,* the trigger consisting of the set of T_{wid} rare nets, is activated by any test patterns in TP. To that end, we need to (i) construct the trigger, (ii) append it to the original design, and (iii) simulate the design using all test patterns in TP. Performing these steps is computationally expensive. Asymptotically, the time complexity of simulating a design having g gates with p test patterns is $O(gp)$. Although theoretically, the complexity is linear, practically, each reward computation takes several seconds even for medium-sized designs (e.g., for MIPS ≈ 25,000 gates, the rate is ≈ 13 seconds/reward computation), and RL agents require several thousand episodes for convergence [49]. Hence, the training time required for such an approach is impractical for designs with tight time-to-market constraints in the semiconductor industry.

Solution 1. To ① avoid reliance on test patterns and ② reduce the time required to compute the reward, we modify the training process by characterizing the target design (before training) and saving the intermediate results. The intermediate results are used during training for computing the reward quickly. Next, we explain how we refine the preliminary formulation by modeling the goal of an adversary using *offline characterization.*

To increase the likelihood of evading unknown test patterns, an adversary needs to *identify combinations of rare nets that are compatible but are least likely to be activated simultaneously.* However, identifying combinations of rare nets is non-trivial as the search space is huge: assuming N rare nets in a design, and T_{wid} being four, there are ${}^{N}C_4$ combinations, which increases combinatorially with the number of rare nets. For example, MIPS has 1,005 rare nets leading to 4.2×10^{10} different triggers. The problem is further exacerbated because picking a set of rarest T_{wid} nets might not work since (i) they are not compatible (*i.e.,* the constructed HT trigger is invalid as it can never be activated), or (ii) they are very likely to be activated by detection techniques. Thus, to estimate the likelihood of compatible rare nets being activated simultaneously, we leverage a randomized algorithm, which we describe next.

The algorithm runs for T iterations and maintains $T + 1$ sets, $C_1, C_2, \ldots, C_T$ (C_i denotes the compatible rare nets in the i^{th} iteration), and a set of unexplored rare nets $\mathcal{U}$. At the start of i^{th} iteration, C_i and $\mathcal{U}$ are initialized as an empty set and as a set of all rare nets in the design, respectively. Next, a random rare net, r_j, is selected from $\mathcal{U}$. If r_j is compatible with C_i, it is removed from $\mathcal{U}$ and added to C_i; otherwise, r_j is just removed from $\mathcal{U}$. This process is repeated until $\mathcal{U}$ is empty, *i.e., N* times. Then, the algorithm starts the next iteration, $i + 1$.[6] At the end of i^{th} iteration, we get C_i, a set of compatible rare nets. These C_i's, $\forall i \in \{1, 2, \ldots, T\}$, are used to compute the likelihood of a given set of compatible rare nets being activated simultaneously, which is translated into a reward for the agent. More specifically, the reward function is updated as

$$\mathcal{R}(s_t, a_t) = \begin{cases} 0, & \text{if } (|s_{t+1}| \neq T_{wid}) \ \wedge \ (\mathcal{I}_C(s_t, a_t) = 1) \\ \rho_1, & \text{if } (|s_{t+1}| \neq T_{wid}) \ \wedge \ (\mathcal{I}_C(s_t, a_t) = 0) \\ 0, & \text{if } (|s_{t+1}| = T_{wid}) \ \wedge \ (\exists\, i \mid s_{t+1} \subseteq C_i) \\ \rho_2, & \text{if } (|s_{t+1}| = T_{wid}) \ \wedge \ (\nexists\, i \mid s_{t+1} \subseteq C_i) \end{cases}$$

Following this reward, the agent tries to select rare nets that are compatible but least likely to be activated simultaneously and, thereby, likely to evade unknown test patterns from detection techniques. Empirical results on designs of different types and sizes validate this hypothesis (§5).

Since the characterization is performed before training, the time complexity of computing rewards using this approach is $O(T)$, where T is the number of sets (C_i) calculated during characterization. The value of T controls the trade-off between runtime and the efficacy of the generated HTs. Larger (smaller) values of T require larger (less) time for characterization and reward computation and generate HTs that are more (less) likely to evade the detection techniques. Note that the theoretical complexity of reward calculation during training is independent of the size (*i.e.,* the number of gates) of the underlying design. In practice, each reward computation takes a few milliseconds, even for large designs such as AES, GPS, and mor1kx with more than 150,000 gates.

Thus, the offline characterization approach helps alleviate the reliance on test patterns from HT detection techniques. It also helps reduce the reward computation time, leading to faster training. Table 2 depicts the training rates (in steps/minute and episodes/minute) for the two approaches for computing rewards: (i) using test patterns during training and (ii) using the information saved during offline characterization. Note that using offline characterization to pre-compute information and using them during training is more than 16× faster than the online calculation approach.

4.4 Trimming: Avoiding Redundant Actions

Challenge ③. Recall that, in the preliminary formulation, the actions available to the agent (at each time step) remain the same irrespective of the state of the agent. This leads to situations where the agent chooses an action that (i) has already been chosen in the past (*i.e.,* a rare net that is already present in the current state s_t) or (ii) is known to not lead to a new state (*i.e.,* a rare net known to be incompatible with at least one of the rare nets in the current state). Choosing such actions makes the training process inefficient as these actions do not lead the agent to a new state. Consequently, the time spent by the agent on such steps is wasted.

Solution 2. To increase the efficiency of the agent in choosing actions while training (thereby reducing the training time), we trim the actions available to the agent based on the state at any given

[6] Note that each iteration of this algorithm is independent. Hence, in our implementation, we parallelize the algorithm.

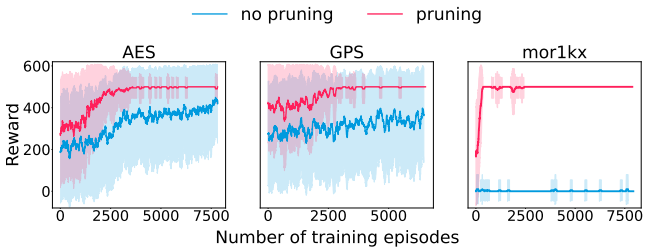

Figure 2: Agent's inability/ability to learn without/with pruning. Total runtime for all plots is 12 hours. The shaded region represents the standard deviation.

time step. Doing so increases the likelihood that at each time step, the agent chooses an action that leads it to a new state.

Ideally, to perform trimming during training, we would like to know which combinations of rare nets are compatible (or not). Therefore, we compute the (in)compatibility of a set of rare nets using a SAT solver. However, if there are N rare nets, we would need to invoke the SAT solver 2^N times (2^N subsets of N rare nets) to compute the compatibility/incompatibility of all possible subsets of rare nets. Since such an approach is infeasible, we use an approximate approach, as explained next.

Instead of computing the compatibility of all subsets of rare nets, we only compute the compatibility of all subsets of size up to S ($<< N$). This compatibility information is computed once and saved for later use.[7] During training, when the episode begins in a random state (*i.e.,* a singleton set with a random rare net), all actions (*i.e.,* rare nets) that are not compatible with the initial state are trimmed off. Then, at each step, after choosing a valid action, the available action list is updated to trim off all rare nets that are not compatible with the latest chosen action.

4.5 Scalability: Pruning Number of Branches

Incorporating solutions 1 and 2 enables the agent to train well and generate stealthy HTs for designs such as `MIPS` ($\approx 25,000$ gates). However, the agent's performance on large designs (`AES`, `GPS`, and `mor1kx` with $> 150,000$ gates) is sub-optimal, as explained next.

Challenge ④. Our updated RL framework (including solutions 1 and 2) is not scalable to large designs such as `AES`, `GPS`, and `mor1kx`. For these designs, the agent is unable to learn an optimal/near-optimal policy even after 12 hours of training (see the blue curve in Figure 2). A possible workaround would be to train for longer time, however, such an approach is not scalable for designs with tight time-to-market constraints in the semiconductor industry.

Challenge ⑤. Another limitation with the training process is that for `mor1kx`, the agent generates only a small number of stealthy HTs. This is due to the structure of the design, as described below. Many rare nets are compatible with several other rare nets making it relatively easier for detection tools to detect HTs. The agent generated only 67 stealthy HTs across 12 hours of training. Ideally, an HT generation framework should generate a large corpus of stealthy HTs, providing an adversary with several options (footnote 5). Next, we outline how we overcome these challenges with a unified solution that aids the agent in choosing better actions.

Solution 3. Recall that we initialize the agent in a random state (*i.e.,* with a random rare net). The agent needs to learn an optimal sequence of decisions starting from each of the rare nets, *i.e.,* the agent needs to learn N branches, where N is the number of rare nets in the design. Since `AES`, `GPS`, and `mor1kx` have a large number of gates, in 12 hours, the agent performs only 7,923, 6,562, and 8,043 episodes (*i.e.,* trials), respectively. Thus, on average, for each branch, the agent only performs 9.27, 7.69, and 12.54 episodes, respectively, which is insufficient for the agent to learn. A simple approach to increase the number of episodes per branch would be to increase the training time. However, this approach is neither feasible nor scalable. A better approach is to reduce, *i.e.,* prune the number of branches for the agent to learn. However, pruning needs to be done carefully so that *the branches that contain stealthy HTs are not pruned off.*

We rely on the data collected during offline characterization to decide which branches (*i.e.,* rare nets as initial states) to prune. We estimate the likelihood of each branch leading to easy-to-detect HTs and choose the branches that are least likely. The likelihood, $L(\cdot)$, of each branch (*i.e.,* rare net, denoted as r_j for $j = 1, 2, \ldots, N$) is estimated as the number of times that rare net is activated in compatible sets, C_i's, computed during characterization. Mathematically, $L(r_j) = \big|\{i \mid r_j \in C_i\}\big|$. B branches with the lowest $L(\cdot)$ are picked, and the other branches are pruned off.

B controls the trade-off between the (ease of) learning of the agent and the quality of generated HTs. Large values of B provide the agent a large space of candidate HTs to choose from. However, the agent would need to learn more branches, requiring more time. Smaller values of B restrict the space from where the agent can choose HTs (*i.e.,* it could potentially miss out on some stealthy HTs from the pruned-off branches), but assuming a fixed training time, the number of episodes that the agent spends on each of the B branches would be larger, allowing better learning. The red curves in Figure 2 showcase the reward trends obtained after implementing pruning. After pruning, the reward curves ramp up quickly, *i.e.,* the agent learns to generate stealthy HTs. Additionally, we observe that with pruning, the standard deviation in the reward values reduces to 0 as the agent learns, whereas, without pruning, even after 12 hours (*i.e.,* $> 6,000$ episodes), either the standard deviation is still very large (≈ 170 for `AES` and ≈ 214 for `GPS`) or the reward is close to 0 (for `mor1kx`). Having overcome all these challenges, we discuss the final architecture of ATTRITION next.

4.6 Final Architecture

Figure 3 illustrates the final architecture of ATTRITION. Given a design, we first identify the rare nets (based on a rareness threshold) using functional simulations. Next, we perform two offline computations: (i) compatibility information of each unique pair of rare nets (for trimming actions) and (ii) likelihood of activation of different combinations of rare nets (*i.e.,* offline characterization). Both computations are parallelized. Then, we begin the training phase. If the design contains less than $\mathcal{G}$ gates, we initiate each

[7] Since the compatibility computation for each unique pair is independent, we parallelize the computation to reduce the runtime. Additionally, since we already generate this compatibility information for trimming actions, we also use it in the offline characterization algorithm (§4.3) when we check if a random rare net r_j from $\mathcal{U}$ is compatible with C_i. This helps reduce the runtime of the characterization algorithm.

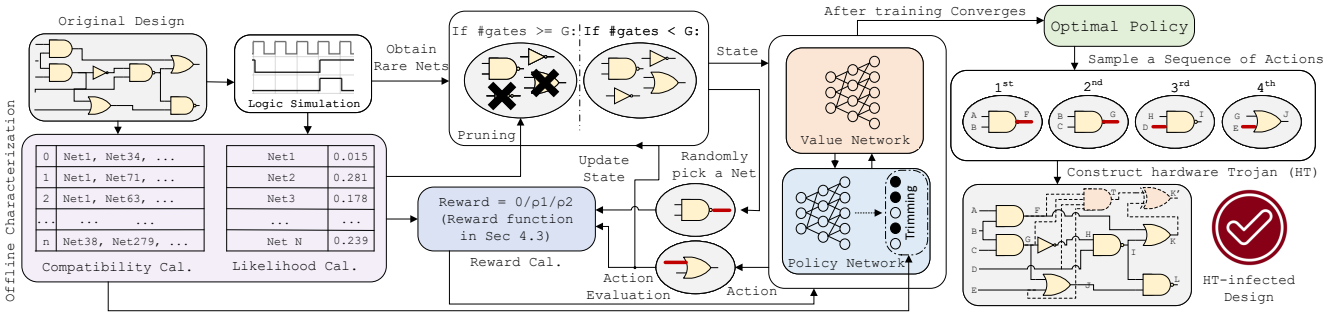

Figure 3: Architecture of ATTRITION.

Table 3: Comparison of HT attack success rate (in percentage; defined as 100 − HT activation rate) against TARMAC for AES, GPS, and mor1kx: Random vs. RL without pruning vs. RL with pruning ($B = 1$).

Design	Randomly-generated HTs	RL-generated HTs		
		No pruning	With pruning	Improv.
AES	48	81	99	**1.22×**
GPS	62	77	95	**1.23×**
mor1kx	1	9	87	**9.66×**

training episode with a random rare net. If the design contains more than or equal to $\mathcal{G}$ gates, we use pruning (§4.5) and initiate each training episode with a randomly picked rare net from the B rare nets that are not pruned off. In both cases, the agent takes action (*i.e.,* chooses a rare net) according to the policy (parameterized by a neural network) and the trimmed action space (*i.e.,* compatible rare nets). The action chosen by the agent is evaluated to compute the reward and to move the agent to the next state. This process repeats $T_{wid} - 1$ times (because the first rare net is already provided to the agent as the initial state), constituting an episode. After a certain number of episodes, the underlying RL algorithm translates the rewards into losses, which are used to update the policy and value neural networks. Eventually, the parameters of the neural networks are tuned, losses become negligible, and reward saturates at the maximum value. Finally, we use the best episodes (*i.e.,* episodes that have the largest rewards after the agent has converged) generated by the agent to create our HTs. ATTRITION (i) is independent of test patterns from HT detection techniques, (ii) has inexpensive reward calculation while training, (iii) selects rare nets effectively, and (iv) is scalable to large designs.

The final architecture described above is independent of detection techniques and test patterns. The objective of the RL agent is not to evade a given set of test patterns; instead, an unknown set of test patterns. It does so by *selecting combinations of trigger nets that are compatible but not likely to be activated simultaneously.* Thus, the HTs generated by ATTRITION are not likely to be activated by the logic testing-based detection techniques (§5.2). On the other hand, side channel-based detection techniques apply test patterns to maximize the impact of the HT on side-channel modalities (*i.e.,* switching activity). To have a measurable impact (beyond the noise level) on switching activity, the HT needs to be activated by the defender [24, 25]. However, since ATTRITION generates HTs that are not likely to be activated, these HTs also evade side channel-based detection techniques, as shown in §5.3. Thus, the HTs generated by ATTRITION evade both logic testing- and side channel-based detection techniques considered in this work, as evidenced by our results in the next section.

5 RESULTS

We now explain our experimental setup and evaluate the performance of ATTRITION in evading state-of-the-art hardware Trojan (HT) detection techniques that are well-cited in academia and published in premier security venues. We showcase the attack success rates against logic testing-based (MERO [20], GA+SAT [21], TGRL [23], TARMAC [22]) and side channel-based (MERS, MERS-h, MERS-s [24], MaxSense [25]) detection techniques. Next, we demonstrate ATTRITION's ability to generate stealthy HTs of different sizes (*i.e.,* trigger widths) followed by showcasing the ramifications of HTs in performing cross-layer attacks.

5.1 Experimental Setup

Implementation Setup. We implement ATTRITION using *PyTorch 1.12* and *stable-baselines3* and train it using a Linux machine with AMD *Ryzen Threadripper PRO 3975WX* with an NVIDIA *A5000* GPU. We use the Proximal Policy Optimization [54] as our reinforcement learning (RL) algorithm. We have a two layered (64 × 64) fully-connected neural network with Tanh activation function for our policy and value networks. We select the reward parameters, ρ_1 and ρ_2, as −1000 and 500, respectively (§4). These values are chosen so that reward is positive only when the agent selects rare nets that are compatible (*i.e.,* would form a valid HT), and the HT is likely to evade the test patterns. We set T, the number of iterations in the offline characterization algorithm (§4.3), equal to the number of test patterns used for the logic testing-based detection techniques. We set the parameter S, which controls the number of compatibility calculations and the runtime for the calculations (§4.4), to be 2 since the number of compatibility calculations for $S = 3$ is much larger. For example, for MIPS with 1005 rare nets, for $S = 2$, the number of calculations is 504, 510 whereas, for $S = 3$, it is $\approx 1.6 \times 10^8$. We set the parameter B, which controls the number of branches we do not prune in §4.5, as 1 because that provides enough HTs for the agent to explore and allows the agent to perform about 7, 000 episodes

Table 4: Comparison of attack success rate (defined as 100−HT activation rate) for random HTs and ATTRITION-generated HTs for logic testing-based detection techniques. All HTs have a trigger width (T_{wid}) of four. All numbers are in percentage.

Design	# gates	# rare nets	Random		MERO [20]		GA+SAT [21]		TGRL [23]		TARMAC [22]			All techniques combined		
			Random HTs	RL HTs	Random HTs	RL HTs	Random HTs	RL HTs	Random HTs	RL HTs	Random HTs	RL HTs	Improv-ement	Random HTs	RL HTs	Improv-ement
c6288	2,416	186	68	77	1	40	1	55	21	48	0	41	>**41**×	0	10	>**10**×
c7552	3,513	282	91	94	61	88	50	98	21	64	0	54	>**54**×	0	44	>**44**×
s13207	1,801	604	97	100	85	99	21	96	97	99	1	71	**71**×	1	69	**69**×
s15850	2,412	649	100	100	92	93	34	89	97	99	2	67	**33.5**×	2	57	**28.5**×
MIPS	23,511	1,005	100	100	100	100	61	64	–	–	1	88	**88**×	1	54	**54**×
AES	161,161	854	100	100	100	100	100	100	–	–	48	99	**2.06**×	48	99	**2.06**×
GPS	193,141	853	100	100	100	100	96	100	–	–	62	95	**1.53**×	62	95	**1.53**×
mor1kx	158,265	641	100	100	99	100	49	100	–	–	1	87	**87**×	1	87	**87**×
Average			94.5	96.38	79.75	90	51.5	88	59	77.5	14.37	75.25	>**47.26**×	14.37	64.37	>**37.01**×

of training in 12 hours. Based on empirical observations, we set the threshold $\mathcal{G}$, which decides whether to employ pruning or not, as 100,000 (§4.6). We provide the hyperparameters and training reward trends in Appendix A.5.

Evaluation Setup. We implement MERO [20], GA+SAT [21], TARMAC [22], MERS [24], MERS-h [24], MERS-s [24], and MaxSense [25] in *Python 3.6*, as they are not publicly available.[8] We set a timeout of 24 hours on all detection techniques for generating the test patterns. We received test patterns for TGRL from the authors of [23]. Additionally, we also generate random patterns for each design. Please note that since ATTRITION is agnostic to the test patterns (§4.3), we only require test patterns from detection techniques to evaluate our generated HTs. We generate random HTs following the procedure outlined in the detection techniques, *i.e.,* we randomly selected the trigger nets from the set of rare nets. We verified the validity of random HTs and ATTRITION generated HTs using a Boolean satisfiability check. We evaluated 100 HTs in both cases and set the trigger width, T_{wid}, (*i.e.,* the number of rare nets constituting the trigger) to be 4. Note that HTs with smaller trigger widths are easier to detect than HTs with larger trigger widths since the probability of activating the trigger is proportional to the product of the probabilities of rare nets being set to their rare values. In our experiments (except §5.4), we set the trigger width as 4 to constrain the adversary. Interested readers may refer to Appendix A.3 for the experimental flow.

Designs. In addition to the designs used by the prior detection techniques (*i.e.,* the ISCAS-85 and ISCAS-89 benchmarks widely used by the hardware security community [5, 7, 26]), we also performed experiments on the MIPS processor from OpenCores [55], AES [56], GPS module from *Common Evaluation Platform* [57], and mor1kx [58]. Since the HT detection techniques assume full-scan access,[9] for sequential designs, we modified the designs accordingly [20–25]. Following prior work, the rareness thresholds are: 0.1 for the ISCAS benchmarks, 9e-4 for MIPS, 7.2e-3 for AES, 4e-3 for GPS, and 1e-4 for mor1lx. These thresholds are chosen to have about 1,000 rare nets to enable a fair comparison with prior work [20–25].

[8]We reached out to the respective authors of the HT detection techniques but we did not receive the test patterns at the time of submission.

[9]Full-scan access enables the defender to control the values of all flip-flops in the designs, representing the best case scenario for the defender.

5.2 Evaluation Against Logic testing Techniques

To evaluate the efficacy of ATTRITION, we define the metric **attack success rate** as the percentage of HTs that evade the detection techniques. Mathematically, it is equivalent to 100 − HT activation rate (§2.2). We compare the attack success rates of randomly generated HTs and ATTRITION generated HTs against logic testing-based detection techniques in Table 4.

Since we obtain the TGRL test patterns from [23], we use the same number of test patterns for other logic testing-based HT detection techniques to enable a fair comparison. However, the test patterns did not match the designs for s13207 and s15850. Hence, the success rates of randomly generated HTs are high for these designs. Also, since TGRL does not show results for MIPS, AES, GPS, and mor1kx [23], the corresponding cells in Table 4 are empty.

Our results demonstrate the attack success rates of ATTRITION generated HTs to be significantly higher than randomly generated HTs, *i.e.,* our HTs are **47.26**× more successful in evading the state-of-the-art logic-testing technique, TARMAC [22]. On average, ATTRITION generated HTs achieve a success rate of 96.38%, 90%, 88%, 77.5%, and 75.25% against test patterns generated from random, MERO [20], GA+SAT [21], TGRL [23], and TARMAC [22], respectively. In contrast, the success rates of randomly generated HTs are 94.5%, 79.75%, 51.5%, 59%, and 14.37%, respectively.

Additionally, since in realistic scenarios, the defender is not constrained to rely on a single detection technique, we also evaluate the attack success rates against test patterns combined with all detection techniques considered in this work. Even in such a scenario, our attack success rates are **37**× higher than the success rates of randomly generated HTs. Hence, our results corroborate that the HTs used by prior detection techniques [20–23] provide a false sense of security to the designers. Under the assumption of randomly generated HTs, the state-of-the-art detection technique (TARMAC [22]) detects 85% of the HTs; however, against a motivated adversary (such as ours), TARMAC detects only 24% of the HTs. **Our results highlight that we can successfully insert stealthy HTs even when a defender uses all logic testing-based detection techniques considered in this work.**

Table 5: Comparison of side-channel sensitivities (lower value denotes more successful attack) for random HTs and ATTRITION-generated HTs for side channel-based detection techniques. All HTs have a trigger width of four. All numbers are in percentage.

Design	Random		MERS [24]		MERS-h [24]		MERS-s [24]		MaxSense [25]			All techniques combined		
	Random HTs	RL HTs	Random HTs	RL HTs	Random HTs	RL HTs	Random HTs	RL HTs	Random HTs	RL HTs	Improv-ement	Random HTs	RL HTs	Improv-ement
c6288	1.13	0.71	2.26	1.19	5.78	2.61	6.08	2.85	19.19	6.50	**2.95**×	19.21	7.18	**2.67**×
c7552	0.21	0.23	0.48	0.34	0.56	0.41	0.83	0.58	57.25	16.83	**3.40**×	57.25	16.83	**3.40**×
s13207	0.40	0.51	0.56	0.59	0.64	0.64	0.67	0.68	49.30	10.27	**4.79**×	49.30	10.27	**4.79**×
s15850	0.29	0.32	0.38	0.40	0.42	0.44	0.44	0.47	47.55	13.24	**3.59**×	47.55	13.24	**3.59**×
MIPS	0.02	0.02	0.03	0.03	0.03	0.03	0.03	0.02	2.12	2.48	**0.85**×	2.12	2.48	**0.85**×
AES	5.74e-3	4.97e-3	3.27e-3	4.09e-3	3.29e-3	4.12e-3	3.10e-3	3.65e-3	40.91	0.02	**1667.80**×	40.91	0.02	**1667.80**×
GPS	3.82e-3	4.08e-3	3.34e-3	2.14e-3	3.23e-3	2.10e-3	2.82e-3	2.12e-3	0.19	0.04	**4.05**×	0.19	0.04	**4.04**×
mor1kx	2.48e-3	2.80e-3	1.90e-3	0.91e-3	1.78e-3	0.98e-3	1.90e-3	0.91e-3	1.19	2.79	**0.42**×	1.19	2.79	**0.42**×
Average	0.26	0.22	0.46	0.32	0.93	0.51	1.01	0.57	27.21	6.52	**210.98**×	27.21	6.61	**210.94**×

Table 6: Attack success rate (%) for HTs generated by [59] against TARMAC [22].

Design	c432	c880	c1355	c1908	c3540	c6288	Average
Success Rate	0	0	0.02	0.15	0	0.02	0.03

Table 7: Comparison of percentage of HTs with side-channel sensitivity less than 10%, for randomly generated and ATTRITION generated HTs against MaxSense [25].

Design	c6288	c7552	s13207	s15850	MIPS	AES	GPS	mor1kx	Average
Random HTs	27	19	5	13	100	0	100	99	45.37
RL HTs	81	71	60	60	97	100	100	97	83.25
Improv.	3.00×	3.73×	12×	4.61×	0.97×	>100×	1×	0.97×	>**15.78**×

Recently, Sarihi *et al.* [59] proposed a technique that utilizes RL to insert HTs. However, there are several fundamental differences between our work and [59], which are elucidated in §6.1. We obtained the HT-inserted netlists from the authors of [59] and evaluated the efficacy of their HTs through test patterns generated by TARMAC. As shown in Table 6 the attack success rates of their HTs are close to 0%.[10] In comparison, ATTRITION-generated HTs have a success rate of 58.25% for designs of similar sizes (<7K gates).

5.3 Evaluation Against Side-channel Techniques

We evaluate the success rates of HTs generated by ATTRITION in evading the side channel-based detection techniques in Table 5. To that end, we compare the side-channel sensitivity (a metric chosen by prior [24] and state-of-the-art side channel-based detection techniques [25]) of randomly generated HTs with ATTRITION-generated HTs in Table 5. Recall that side-channel sensitivity measures the amount of switching caused by an HT relative to the switching in an HT-free design (§2). A lower sensitivity value implies that the impact of an HT on the side-channel metrics is overshadowed by the impact of the regular circuit activity and environmental noise.

[10] Appendix A.7 details breakdown of success rates for HTs of different trigger widths.

Table 8: Comparison of sensitivity to power consumption for randomly generated and ATTRITION generated HTs against MaxSense [25].

Design	c6288	c7552	s13207	s15850	MIPS	AES	GPS	mor1kx	Average
Random HTs	18.84	56.32	52.41	53.89	0.23	0.18	0.02	3.83	23.21
RL HTs	6.68	15.97	11.23	16.38	0.06	0.03	0.02	0.28	6.33
Improv. in stealthiness	2.81×	3.52×	4.66×	3.28×	3.55×	5.14×	~ 1×	13.53×	**4.68**×

Therefore, lower values of side-channel sensitivity indicate a more successful attack. On average, the sensitivity of test patterns generated from MaxSense [25] (the state-of-the-art side channel-based technique) for ATTRITION-generated HTs is 210.98× lower than randomly generated HTs. We also perform an experiment to combine the test patterns generated from all detection techniques, in such a scenario, the sensitivity of the combined set of test patterns for ATTRITION-generated HTs is 210.94× lower than randomly generated HTs. As Table 5 and [25] demonstrate, MaxSense is by far the best side channel-based HT detection technique (among the techniques considered in this work), so, in the remainder of this section, we perform further analysis of its efficacy. In particular, we analyze the percentage of HTs evading detection from side channel-based detection techniques based on the threshold (side-channel sensitivity) of 10% used by prior works [25, 60]. An HT is considered detected if the sensitivity is greater than the threshold, *i.e.,* 10%. Table 7 demonstrates the percentage of HTs evading detection from MaxSense [25] (results for other techniques are shown in Appendix A.6). On average, 83% of ATTRITION generated HTs evade detection from MaxSense [25], whereas, for randomly generated HTs (*i.e.,* the evaluation approach used by prior work), only 45% of the HTs evade detection. **Our results highlight that we can successfully insert HTs even when a defender uses all side channel-based detection techniques considered in this work.**

The side channel-based HT detection techniques considered in this work evaluate the efficacy using logic simulation-based sensitivity analysis as we did in Table 5. To further validate the efficacy of ATTRITION, we performed power simulations to compare the

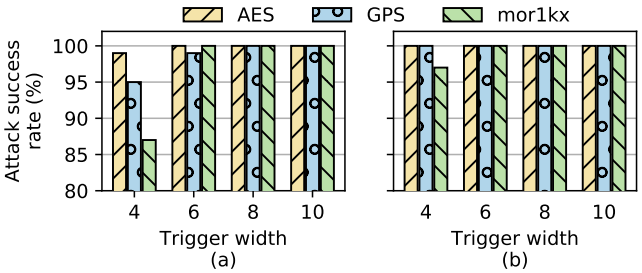

Figure 4: Impact of HT size (*i.e.,* trigger width) on attack success rates against (a) TARMAC [22] and (b) MaxSense [25] for AES, GPS, and mor1kx.

sensitivity of Attrition-generated HTs and randomly generated HTs against MaxSense test patterns. In particular, following the user guides of industrial tools, we executed the industrial tool flow to measure the stealth of Attrition-generated HTs in evading power-based side-channel analysis. Given the HT-inserted netlists and MaxSense test patterns, we perform logic simulations using Synopsys VCS to obtain a value change dump (VCD) that contains information about the switching activity of all nets in the netlists. This VCD and the academic *NanGateOpenCell45nm* library files are provided to Synopsys PrimeTime (a widely-used industrial tool used by semiconductor companies for power simulations to accurately predict the power characteristics of a manufactured chip) to obtain power consumption traces for the MaxSense test patterns.

The power traces for the randomly generated HTs and Attrition-generated HTs are compared with the power trace for the HT-free netlist to calculate the percentage deviations in the power consumption, *i.e.,* sensitivity to power consumption, for the randomly- and Attrition-generated HTs. Table 8 depicts the sensitivity to power consumption for the randomly- and Attrition-generated HTs. On average, the sensitivity of Attrition-generated HTs is 4.68× lower than randomly-generated HTs.

5.4 Impact of Different Sizes of HTs

Now, we analyze the efficacy of Attrition generated HTs as trigger width (T_{wid}) increases. We generate HTs for the largest designs (AES, GPS, and mor1kx) for trigger widths of 4, 6, 8, and 10. Figure 4 (a) illustrates the attack success rates against TARMAC [22], the state-of-the-art technique in logic testing-based detection.[11] As the trigger width increases, Attrition generated HTs evade TARMAC [22] (100% attack success rate) for all the designs.

We observe similar results against MaxSense (Figure 4(b)). As the trigger width increases, the attack success rate increases. At first glance, this result might seem counter-intuitive since an HT with larger trigger width requires more logic gates, which means that the impact of the HT on switching would be larger. However, since our HTs are stealthy, they are not activated by the test patterns and hence have no impact on switching, leading to a success rate of 100% for larger trigger widths.

[11]We do not perform experiments for MERO [20] and GA+SAT [21] as our attack success rate against these techniques is already 100% for a trigger width of 4.

Table 9: Comparison of attack success rate against TARMAC [22] for Attrition minus RL, Synopsys TestMAX [61], and Attrition.

Design	c6288	c7552	s13207	s15850	MIPS	AES	GPS	mor1kx	Average
Attrition minus RL	51	33	59	37	25	72	80	40	49.62
Impr./ Attrition minus RL	0.80×	1.63×	1.20×	1.81×	3.52×	1.37×	1.18×	2.71×	**1.71**×
Synopsys TestMAX	0	0	2	2	2	39	59	1	13.12
Impr./Synopsys TestMAX	>41×	>54×	35.5×	33.5×	44×	2.53×	1.61×	87×	>**37.39**×

5.5 Is Reinforcement Learning Necessary?

Till now, our results demonstrated that Attrition (consisting of domain-specific optimizations and RL agent) generates stealthy HTs that evade the detection techniques considered in this work. Here, we showcase that RL is an integral part of Attrition and crucial to generating stealthy HTs. To that end, we develop a new HT-insertion algorithm called "Attrition minus RL." In other words, we use the same compatibility information, trimming, and detection metrics as Attrition, but generate HTs *without* using an RL agent. In particular, we generate HTs iteratively. In each iteration, a set of trigger nets are chosen and evaluated using the characterization, compatibility information, and trimming outlined in §4.3 and §4.4. If trigger nets are (i) not compatible or (ii) activated by any of the sets (C_i) from offline characterization, they are discarded. Otherwise, we generate an HT. To enable fair comparison, this algorithm is executed for the same duration as the RL training. We compare the success rates of Attrition minus RL and Attrition against TARMAC in Table 9.

Attrition minus RL achieves an average success rate of 49.62% (Table 9) whereas Attrition has a success rate of 75.25% against TARMAC (Table 4). Moreover Attrition has a success rate of 1.71× that of Attrition minus RL, showcasing that utilizing domain-specific optimizations are not sufficient and that RL plays a significant role in Attrition toward generating stealthy HTs.

Till now, we evaluated the efficacy of Attrition with regards to randomly generated HTs, which is consistent with the assumptions made in detection techniques considered in this work. To further demonstrate the improvement in stealthiness offered by Attrition, we consider a better approach for inserting HTs, *i.e.,* we generate HTs that are guaranteed to evade test patterns generated by a commercial IC testing tool. To that end, we generate test patterns using Synopsys TestMAX (for a given design) and use a random sampling approach to create a corpus of HTs that evade these test patterns [61].[12] We showcase the success rates of the HTs generated using TestMAX test patterns in Table 9. Results demonstrate the superiority of Attrition-generated HTs; overall, we achieve a success rate of > 37.39× compared to HTs generated to evade TestMAX test patterns.

[12]Since the generated HTs are guaranteed to evade test patterns; this approach is superior to randomly-generated HTs.

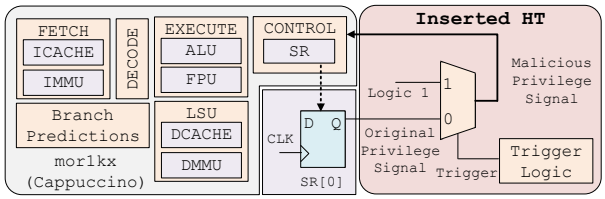

Figure 5: HT performing privilege escalation in `mor1kx`.

Table 10: Instruction sequences to launch a privilege escalation attack and to activate a kill switch (*i.e.,* force the processor in an infinite loop) using Attrition generated HTs for `mor1kx`.

Clock Cycle	Privilege Escalation
1	`l.j 0xc0000`
3	`l.bf 0x3ee7ef4`
6	`l.mtspr r7, r7, 0x0`
9	`l.j -0x123fff0`
12	`l.mulu r15, r15, r15`

Clock Cycle	Kill Switch
1	`l.j -0x4`
3	`l.mtspr r18, r18, 0x0`
6	`l.mtspr r9, r18, 0x1000`
9	`l.jr r2`
12	`l.mtspr r5, r25, 0x4000`
15	`l.mtspr r11, r24, 0x0`
18	`l.mulu r31, r0, r0`

5.6 Ramifications of Stealthy Hardware Trojans

Till now, our results demonstrated that Attrition generates HTs that evade the considered detection techniques; however, this does not necessarily imply that they can be used to launch practical attacks. Here, we demonstrate the usefulness of Attrition generated HTs by showcasing two case-studies that undermine the security of the largest processor considered in this work, *i.e.,* `mor1kx`.

Privilege Escalation. To perform privilege escalation, we need to identify the high-level instructions an adversary should use to trigger the rare nets, *i.e.,* the Boolean values of the rare nets need to be mapped to valid assembly-level instructions. Given Attrition generated rare nets, we write formal properties corresponding to the values of those rare nets and use *Cadence JasperGold* to provide binaries that would trigger the HT. Finally, we translate the binaries into valid instructions using the processor architecture manual [62]. Note that the devised HT that causes privilege escalation evades all detection techniques considered in this work, as confirmed by our experiments.

For `mor1kx`, one rare net chosen by Attrition is from the processor control unit, another is from the system bus, and the other two are from the fetch stage. The target flip-flop to perform privilege escalation is a part of the special-purpose Supervision Register, SR. The least significant bit of SR holds the mode of the processor: 0 indicates user mode and 1 indicates supervisor mode. Table 10 denotes the instruction sequence that needs to be executed to perform privilege escalation. The attack requires 12 clock cycles since the first instruction `l.j 0xc0000` to activate the HT that flips the privilege bit (Figure 5). We illustrate the waveform of the attack execution in Appendix A.9.

Kill switch. Here, we demonstrate a case study on how Attrition generated HTs can be used to force the `mor1kx` processor in an infinite loop, *i.e.,* stop further useful execution when the malicious instructions are executed. Attrition picks the four rare nets as follows: one from the system bus, one from the fetch stage, and two from the processor control unit. The target register is the program counter. Table 10 denotes the instruction sequence that need to be executed to launch the attack. This attack requires 20 clock cycles since the first instruction `l.j -0x4`. Upon HT activation, the program counter gets stuck at `0x00000000` preventing further execution until a restart is performed. Essentially, this HT is a demonstration of a "*kill switch*" that can be activated by executing a handful of malicious instructions.

Thus, these case studies demonstrate that an adversary can repurpose Attrition to design HTs that not only evade detection, but also cause practical, cross-layer, and real-world attacks.

6 RELATED WORK

While this work assumes a hardware Trojan (HT) inserted in an untrusted foundry (the threat model highlighted by the U.S. Department of Defense [4]), we highlight how our work relates to HTs inserted during design time. We demonstrate an additional advantage of Attrition in generating *dynamic* HTs as opposed to the *static* HTs available in TrustHub [46], a popular database used by the hardware security community. Finally, we discuss how RL has been used for benchmarking defenses.

6.1 Hardware Trojan Insertion and Evasion

HTs can be inserted during design time (adversary is the third-party intellectual property provider) or during fabrication time (adversary in the foundry). Next, we highlight some examples of HT insertion in an untrusted foundry. Lin *et al.* [41] designed an HT that induced physical side-channels to leak the keys of an AES cryptographic accelerator. Becker *et al.* [15] modified the dopant polarity of transistors that forces the input of the logic gates to a constant logic level of 0 or 1. Yang *et al.* [12] performed privilege escalation using a capacitor-based HT on fabricated chips.

For HTs inserted during design time, researchers proposed various insertion and corresponding evasion techniques, leading to a *cat-and-mouse* game, which we briefly outline next. [10] identifies unused portions of the circuit during design-time testing and flags them as potentially malicious. They attempt to detect HTs by identifying pairs of dependent signals in the design that could ostensibly be replaced by a wire without impacting the outcome of verification test cases. However, Sturton *et al.* [16] thwarted the detection technique proposed in [10] by developing HTs such that no pair of dependent signals are always equal during design-time testing. Functional analysis for nearly-unused circuit identification (FANCI) identifies input wires that can serve as backdoor triggers by measuring the degree of control an input has on the outputs of a design [11]. In addition, VeriTrust examines verification corners to identify trigger signals [63]. However, DeTrust [64] evades FANCI [11] by spreading the trigger logic across multiple combinational logic blocks, while HT triggers are hidden into multiple sequential levels to deter VeriTrust [64].

In contrast, ATTRITION: (i) is an automated framework to generate stealthy HTs, requiring no manual intervention; (ii) can adapt to new detection techniques and generate HTs dynamically, so, if a new detection technique is developed in the future, our framework would still hold even if the current HTs we generate may or may not; (iii) is not a point solution to defeat a specific detection technique unlike the efforts in the design-time attack model, so, is not subject to a cat-and-mouse game of attacks and defenses.

More recently, Sarihi *et al.* [59] also utilize RL to insert HTs. However, several key differences exist. First, the modeling of states, actions, and rewards of both techniques differ considerably. Second, ATTRITION focuses on inserting HTs that evade detection techniques, whereas [59] insert HTs that maximize the number of inputs (*i.e.,* trigger width). Third, [59] does not evaluate the efficacy of generated HTs in evading detection techniques. In contrast, the prime objective of ATTRITION is generating HTs that evade detection techniques (across logic testing-based and side channel-based approaches). Fourth, ATTRITION incorporates three solutions using domain knowledge, rendering it scalable; such solutions providing experimental evidence of scalability are missing in [59]. Finally, ATTRITION demonstrates results on designs up to >190K gates, whereas the largest design in [59] has only ≈7K gates.

6.2 *"Static"* Hardware Trojan Benchmarks

TrustHub [46, 47] is a database of different HT benchmarks (a total of 91) and provides a common platform for researchers to evaluate detection techniques. However, the HTs in TrustHub are created from randomly selected rare nets, *i.e.,* they do not reflect a real-world adversary because the adversary inserts HTs in a directed manner with the objective of evading detection. Moreover, these benchmarks were last updated in 2017, and since then, there have been several new HT detection techniques. So, the benchmarks do not keep up with the developments in HT detection since they are static, unlike our framework, which can generate HTs targeted towards new detection techniques too.

After TrustHub, there have been several works that have developed automated tools for HT insertion [48, 65, 66]. However, most automated tools for HT insertion rely on signal probability, *i.e.,* the generated HTs have a very low probability of getting triggered during logic testing [66]. Another limitation of these tools is that they insert HTs in the design by randomly picking the rare nets from a pool of rare nets [48, 66]. However, an adversary is not constrained to design the trigger randomly. Instead, an adversary would characterize the design space of the underlying design and insert HTs that are likely to evade many detection techniques. Additionally, the benchmarks from the aforementioned HT insertion tools are "static" in nature, *i.e.,* they have not evolved with time and new detection techniques.

In contrast, ATTRITION generates HTs in an automated manner and can readily adapt to new detection techniques, thereby generating HTs dynamically and providing a litmus test for existing (and future) HT detection techniques. We envision that such a set of HT-infested benchmarks can aid researchers in developing detection techniques that do not provide a "*false sense of security.*"

6.3 Potential Countermeasures

ATTRITION generates stealthy HTs that evade eight detection techniques spanning logic testing and side channel-based approaches (§5). As a result, our work highlights the requirement of developing robust HT detection techniques. Next, we outline two directions that could potentially mitigate the attack presented in this work.

The first direction involves modifying the GDSII of the design to prevent HT insertion. Designers can modify the GDSII by making the design congested [40, 67, 68] or increasing the utilization [40, 67, 68]. Such an approach makes the targeted insertion of HTs difficult for an adversary. Recall that HT insertion entails augmenting logic gates and wires into the underlying design, and if there is a dearth of space (in terms of placement sites and/or routing tracks) in the GDSII, an attacker faces considerable challenges. ICAS took the first step toward this direction [40], and the toolchain enables designers to evaluate the susceptibility of a GDSII toward HT insertion. However, increasing congestion and utilization of the chip is usually impractical since timing closure becomes challenging [67]. Real-world designs such as processors are rarely fabricated with utilization greater than 70% to allow engineering change order-related fixes. This provides enough space for inserting HTs, thereby limiting the efficacy of such defenses.

Another direction is a consequence of our results in Table 4. Consider the design `c6288` for which ATTRITION's success rate is low. As analyzed and explained in Appendix A.5, this is due to the structure of the design where several rare nets are compatible with each other, which increases the ease of activating many rare nets simultaneously, leading to easy detection of HTs. As part of future work, we intend to explore a way to mitigate attacks such as ATTRITION by making changes to the structure of a given design (through synthesis) so that many rare nets are compatible with each other. This way, we can make detecting HTs easier for existing detection tools.

7 CONCLUSION AND RAMIFICATIONS

Hardware Trojans (HTs) inserted during fabrication pose a potent threat to the security of integrated circuits. Unfortunately, we note that state-of-the-art HT detection techniques have been tested only with *weak* adversarial assumptions (*i.e.,* random insertion of HTs), providing a *"false sense of security"* for over a decade. This calls for a critical rethinking of security evaluation methodologies for HT detection techniques.

Aided by the power of reinforcement learning (RL), we developed ATTRITION that automatically generates stealthy HTs that evade eight HT detection techniques from two different approaches, including the state-of-the-art. To that end, we overcame five challenges related to computational complexity, efficiency, scalability, and practicality using three solutions. We showcased the efficacy of ATTRITION on different designs ranging from widely-used academic benchmarks to processors. ATTRITION achieved average attack success rates 47× and 211× compared to randomly inserted HTs against the state-of-the-art logic testing and side channel techniques. Additionally, we illustrated cross-layer attacks through two case studies (privilege escalation and kill switch) on the open-source `mor1kx` processor, demonstrating an end-to-end methodology for

generating HTs that are not only stealthy, but also useful for realistic attacks. As part of future work, we intend to examine how an RL-based defender [69] can spar with an RL-based attacker (such as ATTRITION) to generate interesting attacks and defenses.

Ramifications. ATTRITION calls for action on re-evaluating the assumptions made by the HT detection techniques. Rather than considering randomly inserted HTs, developers of future HT detection techniques must consider the existence of a motivated adversary. To that end, ATTRITION (i) demonstrates the reduction in the efficacy of HT detection techniques when ATTRITION generated HTs are considered, (ii) aids in creating a suite of HT-infested designs that can be used by researchers to benchmark the strength of their detection techniques, and (iii) urges the community to develop efficient detection techniques that are scalable to real-world designs.

ACKNOWLEDGMENTS

The work was partially supported by the National Science Foundation (NSF CNS–1822848 and NSF DGE–2039610). We thank Amin Sarihi and Prof. Abdel-Hameed A. Badawy from New Mexico State University for providing the HT designs of their work. In addition, we thank Prof. Dileep Kalathil, Zaiyan Xu, Rahul Kande, Chen Chen, and the anonymous reviewers for their constructive comments.

REFERENCES

[1] Ramish Zafar. TSMC's Total 3nm Investment Will Equal At Least $ 23 Billion. https://wccftech.com/tsmc-3nm-investment-23-billion-project-end/, 2021. [Online; last accessed 2-May-2022].

[2] Yang Jie. TSMC to Invest Up to $44 Billion in 2022 to Beef Up Chip Production. https://www.wsj.com/articles/tsmc-to-invest-up-to-44-billion-in-2022-to-beef-up-chip-production-11642076019, 2022. [Online; last accessed 2-May-2022].

[3] Mark Hachman. Chip shortages will continue until 2023. https://www.pcworld.idg.com.au/article/687673/chip-shortages-will-continue-until-2023-superfoundry-tsmc-says/, 2021. [Online; last accessed 2-May-2022].

[4] Department of Defense (DoD). Securing Defense-Critical Supply Chains. *An action plan developed in response to President Biden's Executive Order 14017*, February 2022. [Online; last accessed 2-May-2022].

[5] Yousra Alkabani and Farinaz Koushanfar. Active Hardware Metering for Intellectual Property Protection and Security. In *16th USENIX Security Symposium (USENIX Security 07)*, volume 20, pages 1–20, 2007.

[6] Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh Tripunitara. Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 495–510, 2013.

[7] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan Rajendran, and Ozgur Sinanoglu. Provably-Secure Logic Locking: From Theory to Practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1601–1618, 2017.

[8] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan detection using IC fingerprinting. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 296–310. IEEE, 2007.

[9] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. A Primer on Hardware Security: Models, Methods, and Metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.

[10] Matthew Hicks, Murph Finnicum, Samuel T King, Milo MK Martin, and Jonathan M Smith. Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically. In *2010 IEEE Symposium on Security and Privacy (SP'10)*, pages 159–172. IEEE, 2010.

[11] Adam Waksman, Matthew Suozzo, and Simha Sethumadhavan. FANCI: Identification of Stealthy Malicious Logic Using Boolean Functional Analysis. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, pages 697–708, 2013.

[12] Kaiyuan Yang, Matthew Hicks, Qing Dong, Todd Austin, and Dennis Sylvester. A2: Analog Malicious Hardware. In *2016 IEEE Symposium on Security and Privacy (SP'16)*, pages 18–37. IEEE, 2016.

[13] Timothy Trippel, Kang G Shin, Kevin B Bush, and Matthew Hicks. Bomberman: Defining and Defeating Hardware Ticking Timebombs at Design-time. In *2021 IEEE Symposium on Security and Privacy (SP'21)*, pages 970–986. IEEE, 2021.

[14] Sergei Skorobogatov and Christopher Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 23–40. Springer, 2012.

[15] Georg T Becker, Francesco Regazzoni, Christof Paar, and Wayne P Burleson. Stealthy Dopant-Level Hardware Trojans. In *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pages 197–214. Springer, 2013.

[16] Cynthia Sturton, Matthew Hicks, David Wagner, and Samuel T King. Defeating UCI: Building stealthy and malicious hardware. In *2011 IEEE Symposium on Security and Privacy (SP'11)*, pages 64–77. IEEE, 2011.

[17] Sally Adee. The hunt for the kill switch. *iEEE SpEctrum*, 45(5):34–39, 2008.

[18] Keith Rebello. Safeguards against Hidden Effects and Anomalous Trojans in Hardware (SHEATH). https://www.darpa.mil/program/safeguards-against-hidden-effects-and-anomalous-trojans-in-hardware. [Online; last accessed 17-April-2022].

[19] DARPA Public Affairs. DARPA Selects Teams to Increase Security of Semiconductor Supply Chain. https://www.darpa.mil/news-events/2020-05-27, 2020. [Online; last accessed 2-May-2022].

[20] Rajat Subhra Chakraborty, Francis Wolff, Somnath Paul, Christos Papachristou, and Swarup Bhunia. MERO: A statistical approach for hardware Trojan detection. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 396–410. Springer, 2009.

[21] Sayandeep Saha, Rajat Subhra Chakraborty, Srinivasa Shashank Nuthakki, Debdeep Mukhopadhyay, et al. Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 577–596. Springer, 2015.

[22] Yangdi Lyu and Prabhat Mishra. Scalable Activation of Rare Triggers in Hardware Trojans by Repeated Maximal Clique Sampling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 40(7):1287–1300, 2021.

[23] Zhixin Pan and Prabhat Mishra. Automated test generation for hardware Trojan detection using reinforcement learning. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 408–413, 2021.

[24] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. MERS: statistical test generation for side-channel analysis based Trojan detection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 130–141, 2016.

[25] Yangdi Lyu and Prabhat Mishra. MaxSense: Side-channel Sensitivity Maximization for Trojan Detection Using Statistical Test Patterns. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 26(3):1–21, 2021.

[26] Mohamed El Massad, Siddharth Garg, and Mahesh V Tripunitara. Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes. In *The Network and Distributed System Security Symposium (NDSS)*, pages 1–14, 2015.

[27] Yuan Cao, Chip-Hong Chang, and Shoushun Chen. A cluster-based distributed active current sensing circuit for hardware Trojan detection. *IEEE Transactions on Information Forensics and Security (TIFS)*, 9(12):2220–2231, 2014.

[28] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan Detection using IC Fingerprinting. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 296–310. IEEE, 2007.

[29] Hassan Salmani and Mohammad Tehranipoor. Layout-aware switching activity localization to enhance hardware Trojan detection. *IEEE Transactions on Information Forensics and Security (TIFS)*, 7(1):76–87, 2011.

[30] Dongdong Du, Seetharam Narasimhan, Rajat Subhra Chakraborty, and Swarup Bhunia. Self-referencing: A scalable side-channel approach for hardware Trojan detection. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 173–187. Springer, 2010.

[31] Yier Jin and Yiorgos Makris. Hardware Trojan detection using path delay fingerprint. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 51–57. IEEE, 2008.

[32] Hong Zhao, Luke Kwiat, Kevin A Kwiat, Charles A Kamhoua, and Laurent Njilla. Applying chaos theory for runtime hardware Trojan monitoring and detection. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 17(4):716–729, 2018.

[33] Jiaji He, Yiqiang Zhao, Xiaolong Guo, and Yier Jin. Hardware Trojan Detection Through Chip-Free Electromagnetic Side-Channel Statistical Analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2939–2948, 2017.

[34] Konstantin Böttinger, Patrice Godefroid, and Rishabh Singh. Deep reinforcement fuzzing. In *2018 IEEE Symposium on Security and Privacy workshops*, pages 116–122. IEEE, 2018.

[35] Daimeng Wang et al. SyzVegas: Beating Kernel Fuzzing Odds with Reinforcement Learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2741–2758, 2021.

[36] Xing Liu, Wei Yu, Fan Liang, David Griffith, and Nada Golmie. On deep reinforcement learning security for Industrial Internet of Things. *Computer Communications*, 168:20–32, 2021.

[37] Liang Xiao, Xiaoyue Wan, Xiaozhen Lu, Yanyong Zhang, and Di Wu. IoT security techniques based on machine learning: How do IoT devices use AI to enhance security? *IEEE Signal Processing Magazine*, 35(5):41–49, 2018.

[38] Thanh Thi Nguyen and Vijay Janapa Reddi. Deep Reinforcement Learning for Cyber Security. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, pages 1–17, 2021.

[39] Richard Elderman, Leon JJ Pater, Albert S Thie, Madalina M Drugan, and Marco A Wiering. Adversarial Reinforcement Learning in a Cyber Security Simulation. In *International Conference on Agents and Artificial Intelligence (ICAART) (2)*, pages 559–566, 2017.

[40] Timothy Trippel, Kang G Shin, Kevin B Bush, and Matthew Hicks. ICAS: an extensible framework for estimating the susceptibility of ic layouts to additive trojans. In *2020 IEEE Symposium on Security and Privacy (SP'20)*, pages 1742–1759. IEEE, 2020.

[41] Lang Lin, Markus Kasper, Tim Güneysu, Christof Paar, and Wayne Burleson. Trojan side-channels: Lightweight hardware trojans through side-channel engineering. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 382–395. Springer, 2009.

[42] Alex Baumgarten, Michael Steffen, Matthew Clausman, and Joseph Zambreno. A case study in hardware Trojan design and implementation. *International Journal of Information Security*, 10(1):1–14, 2011.

[43] Prabhat Kumar Mishra and Yangdi Lyu. Trigger activation by repeated maximal clique sampling, January 7 2021. US Patent App. 16/893,701.

[44] Sheng Wei and Miodrag Potkonjak. Scalable hardware Trojan diagnosis. *IEEE Transactions on very large scale integration (VLSI) systems*, 20(6):1049–1057, 2011.

[45] Mainak Banga and Michael S Hsiao. A region based approach for the identification of hardware Trojans. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 40–47. IEEE, 2008.

[46] Trust-Hub. https://www.trust-hub.org/. [Online; last accessed 2-May-2022].

[47] Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. On design vulnerability analysis and trust benchmarks development. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 471–474. IEEE, 2013.

[48] Shichao Yu, Weiqiang Liu, and Maire O'Neill. An improved automatic hardware Trojan generation platform. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 302–307. IEEE, 2019.

[49] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[50] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[51] Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz. Reverse-Engineering a Cryptographic RFID Tag. In *17th USENIX Security Symposium (USENIX Security 08)*, volume 28, 2008.

[52] TechInsights, "Reverse engineering". https://www.techinsights.com/analysis-solutions/reverse-engineering. [Online; last accessed 2-May-2022].

[53] Degate. https://degate.readthedocs.io/en/latest/. [Online; last accessed 2-May-2022].

[54] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[55] OpenCores. Educational 16-bit MIPS Processor. https://opencores.org/projects/mips_16, 2013.

[56] Timothy Trippel. AES. https://github.com/timothytrippel/bomberman/tree/master/third_party/aes_128, 2020. [Online; last accessed 2-May-2022].

[57] MIT Lincoln Laboratory. GPS code generator. https://github.com/mit-ll/CEP/tree/master/hdl_cores/gps, 2021. [Online; last accessed 2-May-2022].

[58] Stafford Horne. mor1kx - an OpenRISC processor IP core. https://github.com/openrisc/mor1kx, 2022. [Online; last accessed 2-May-2022].

[59] Amin Sarihi, Ahmad Patooghy, Peter Jamieson, and Abdel-Hameed A Badawy. Hardware trojan insertion using reinforcement learning. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, pages 139–142, 2022.

[60] Bharathan Balaji, John McCullough, Rajesh K Gupta, and Yuvraj Agarwal. Accurate characterization of the variability in power consumption in modern mobile processors. In *2012 Workshop on Power-Aware Computing and Systems (HotPower 12)*, 2012.

[61] Synopsys. TestMAX ATPG and TestMAX Diagnosis User Guide. Version S-2021.06-SP1, 2021.

[62] Damjan Lampret, Chen-Min Chen, Marko Mlinar, Johan Rydberg, Matan Ziv-Av, Chris Ziomkowski, and et al. OpenRISC 1000Architecture Manual. https://raw.githubusercontent.com/openrisc/doc/master/openrisc-arch-1.4-rev0.pdf, 2022. [Online; last accessed 2-May-2022].

[63] Jie Zhang, Feng Yuan, Linxiao Wei, Yannan Liu, and Qiang Xu. VeriTrust: Verification for hardware trust. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 34(7):1148–1161, 2015.

[64] Jie Zhang, Feng Yuan, and Qiang Xu. Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 153–166, 2014.

[65] Bicky Shakya, Tony He, Hassan Salmani, Domenic Forte, Swarup Bhunia, and Mark Tehranipoor. Benchmarking of hardware trojans and maliciously affected circuits. *Journal of Hardware and Systems Security*, 1(1):85–102, 2017.

[66] Jonathan Cruz, Yuanwen Huang, Prabhat Mishra, and Swarup Bhunia. An automated configurable Trojan insertion framework for dynamic trust benchmarks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1598–1603. IEEE, 2018.

[67] Johann Knechtel, Jayanth Gopinath, Mohammed Ashraf, Jitendra Bhandari, Ozgur Sinanoglu, and Ramesh Karri. Benchmarking security closure of physical layouts: Ispd 2022 contest. In *Proceedings of the 2022 International Symposium on Physical Design*, pages 221–228, 2022.

[68] Ronald P Cocchi, James P Baukus, Lap Wai Chow, and Bryan J Wang. Circuit camouflage integration for hardware ip protection. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–5. IEEE, 2014.

[69] Vasudev Gohil, Satwik Patnaik, Hao Guo, Dileep Kalathil, and Jeyavijayan Rajendran. DETERRENT: Detecting Trojans using Reinforcement Learning. In *Proc. ACM Design Automation Conference (DAC)*, 2022.

[70] Kan Xiao, Domenic Forte, Yier Jin, Ramesh Karri, Swarup Bhunia, and Mohammad Tehranipoor. Hardware Trojans: Lessons learned after one decade of research. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(1):1–23, 2016.

[71] Swarup Bhunia, Michael S. Hsiao, Mainak Banga, and Seetharam Narasimhan. Hardware Trojan Attacks: Threat Analysis and Countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, 2014.

[72] Jie Li and John Lach. At-speed delay characterization for IC authentication and Trojan horse detection. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 8–14. IEEE, 2008.

[73] Domenic Forte, Chongxi Bao, and Ankur Srivastava. Temperature tracking: An innovative run-time approach for hardware Trojan detection. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 532–539. IEEE, 2013.

[74] Seetharam Narasimhan, Xinmu Wang, Dongdong Du, Rajat Subhra Chakraborty, and Swarup Bhunia. TeSR: A robust temporal self-referencing approach for hardware Trojan detection. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 71–74. IEEE, 2011.

[75] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. Structural Attack against Graph Based Android Malware Detection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3218–3235, 2021.

[76] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. Mab-malware: A reinforcement learning framework for attacking static malware classifiers. *2022 ACM Asia Conference on Computer and Communications Security, to appear*, 2022.

[77] Lan Zhang, Peng Liu, Yoonho Choi, and Ping Chen. Semantics-preserving Reinforcement Learning Attack Against Graph Neural Networks for Malware Detection. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2022.

[78] Long Li, Yu Luo, Jing Yang, and Lina Pu. Reinforcement Learning Enabled Intelligent Energy Attack in Green IoT Networks. *IEEE Transactions on Information Forensics and Security (TIFS)*, 17:644–658, 2022.

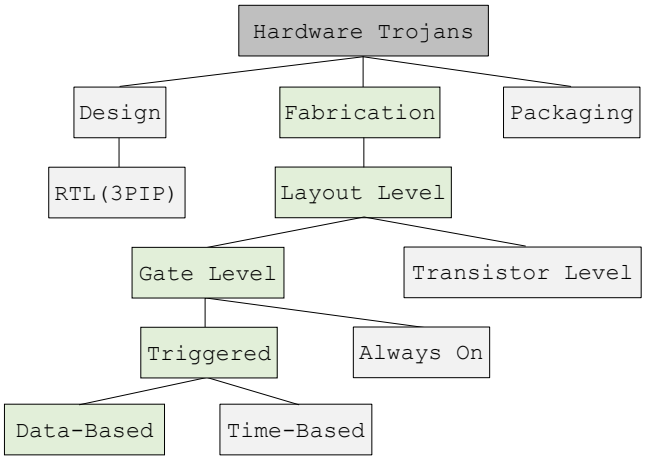

Figure 6: Hardware Trojan taxonomy [13, 70, 71].

A APPENDIX

A.1 Integrated Circuit Design Flow

Developing a complex integrated circuit (IC) involves several design stages heavily assisted by computer-aided design (CAD) tools. First, the high-level design specifications are decided based on the functionality of the intended chip, and designers (or product architects) develop the optimal architecture using the hardware description language (HDL) of choice (e.g., Verilog) at the register-transfer level (RTL). At times, the design team also deploys third-party intellectual property (IP) cores in the design. Next, CAD tools synthesize the HDL codes into a gate-level netlist for a given technology node. After synthesis, the design is taken through floorplanning, placement, and routing through a physical synthesis tool (commonly known as layout generation tools) to create a physical layout. The physical layout is encoded into a graphics database system format which is then sent to a fabrication facility (a.k.a foundry) for fabrication. Once the design is fabricated, it is subject to functional and performance testing. Once the fabricated chips pass the intended tests, they are packaged and sold to end-users. We illustrate the IC design flow and the supply chain in Figure 11. Our work considers the design house and the testing and packaging facility to be trustworthy while the foundry is untrustworthy.

A.2 Hardware Trojan Taxonomy

To better understand hardware Trojans (HTs) and generate effective defense techniques, we need to understand the taxonomy of HTs [70, 71] (Figure 6). We classify HTs based on three attributes: (i) insertion phase, (ii) abstraction level, and (iii) activation mechanism. The insertion phase denotes where an HT can be inserted by an adversary in the IC supply chain, including design, fabrication, and packaging. Since we consider fabrication-time HTs, *i.e.,* HTs inserted by an adversary during the fabrication of ICs, two options emerge. Either an adversary can insert an HT by adding extra logic gates or performing stealthy manipulations on the transistors. In our work, we consider gate-level HTs consistent with most prior works in HT research. Finally, we consider the activation mechanism of HTs, *i.e.,* how an HT will be activated in-field. The triggering mechanisms include always-on and triggered. Always-on HTs have constant-time effects on the chip operation as long as the power is supplied. Triggered HTs are activated through input patterns. External stimulation can be classified into time-based and data-based. Time-based stimulus usually is controlled and applied by counters. Data-based stimulus refers to a single or a sequence of input data patterns.

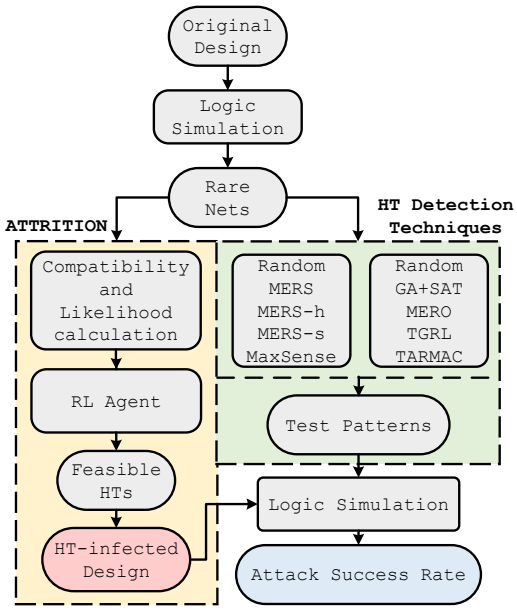

Figure 7: A high-level overview of the experimental flow.

A.3 Experimental Flow

Figure 7 provides a high-level overview regarding our experimental flow. First, we start with an HT-free design. We perform logic simulations on the design to obtain the probabilities of each wire in the design being 0 or 1. Then, according to a pre-determined **rareness threshold**, we obtain the **rare nets** (*i.e.,* nets whose probability of being 0 or 1 is lower than the rareness threshold) from the design. Finally, our RL agent uses these rare nets to generate stealthy HTs. To do so, we first perform the compatibility and likelihood calculations. Then, we train the RL agent, and the HTs are generated once the agent training ends. These HTs are then inserted into the HT-free design to generate HT-infected designs. On the other hand, the rare nets are used by both categories (logic testing- and side channel-based) of detection techniques. Next, we generate test patterns from the detection techniques to detect HTs. Finally, we simulate the HT-infected designs with these test patterns to evaluate the success rate of our attack.

A.4 Resilience of ATTRITION generated HTs

Designers can potentially defend chips against fabrication-time HTs using (i) integrated circuit (IC) fingerprinting [8] and (ii) on-chip sensors [72, 73].

Table 11: Hyperparameters for ATTRITION.

Parameter	Value
Learning Rate	$\begin{cases} 0.0003, & \text{if } (\#\text{ gates} < 100K) \\ 0.003, & \text{if } (\#\text{ gates} \geq 100K) \wedge (\#\text{ timesteps} < 10K) \\ 0.0003, & \text{if } (\#\text{ gates} \geq 100K) \wedge (\#\text{ timesteps} \geq 10K) \end{cases}$
Optimizer	Adam
Max. Num. of Timesteps	$100K$
Num. of Epochs	10
Num. of Steps	50
Batch Size	64
Discount Factor	0.99
Clipping Value	0.2

IC fingerprinting uses side-channel information (e.g., path delay, switching power, etc.) to model the behavior of a golden (*i.e.,* HT-free) IC. Input patterns are applied to HT-free and HT-infested ICs, and measurements are collected. If the difference between the collected measurements exceeds a certain threshold (as determined by the defender), the chip is labeled untrustworthy. Researchers in [8] state that their approach requires the HTs to be at least 0.01% of the circuit—ATTRITION generated HTs occupy (a maximum) of 0.006% of the overall chip area, thus, they cannot be detected.

Note that fingerprinting-based techniques suffer from a significant limitation, *i.e.,* the availability and reliance on a golden, trusted IC. As a result, researchers developed approaches that replace the golden chip with self-referencing [74]. For example, temporal self-referencing [74] compares the transient current of an IC with itself across different time intervals. Unfortunately, the ability of temporal self-referencing (in HT detection) is contingent on finding input patterns that activate the HT. MERS [24] is one of the techniques for generating such input patterns. However, as our results show, ATTRITION generated HTs have minimal impact on switching; hence, they are resistant to self referencing-based methods.

Adding on-chip sensors such as temperature/thermal sensors can also detect fabrication-time HTs [73]. These sensors detect deviations in power/thermal profiles caused by HT activation. Recall that our ATTRITION generated HTs require a minimum of 6 to 10 logic gates for a circuit with $> 150,000$ gates.

A.5 Hyperparameters and Reward Trends

For an interested reader, we demonstrate additional information here. Table 11 lists the hyperparameters of our agent. Figure 8 shows the reward trends for all benchmarks. We observe that for all benchmarks except `c6288`, the reward curves ramp up to the maximum value quickly, indicating that the agent has learned to generate stealthy HTs. The reason for the different reward curve for `c6288` is due to the structure of the design. `c6288`, a 16×16 multiplier, has several rare nets compatible with each other. Hence, it is quite easy for a test pattern to activate several rare nets simultaneously, leading to the agent's inability to find stealthy HTs.

A.6 Additional Results for Side channel-based Detection Techniques

Table 14 shows the percentage of HTs that evade random, MERS, MERS-h, and MERS-s detection techniques, *i.e.,* the percentage of HTs with side-channel sensitivity less than 10%. A majority of HTs (both randomly generated and ATTRITION generated) evade all of the above techniques.

A.7 Additional Results for Efficacy of Related Work on Inserting Hardware Trojans using Reinforcement Learning

Table 13 shows the attack success rates (against TARMAC) of HTs of different trigger widths from [59]. The results indicate that for all benchmarks, all HTs with trigger width less than 5 are detected by TARMAC, and only a tiny fraction of HTs (0.15%) with trigger width 5 evade detection.

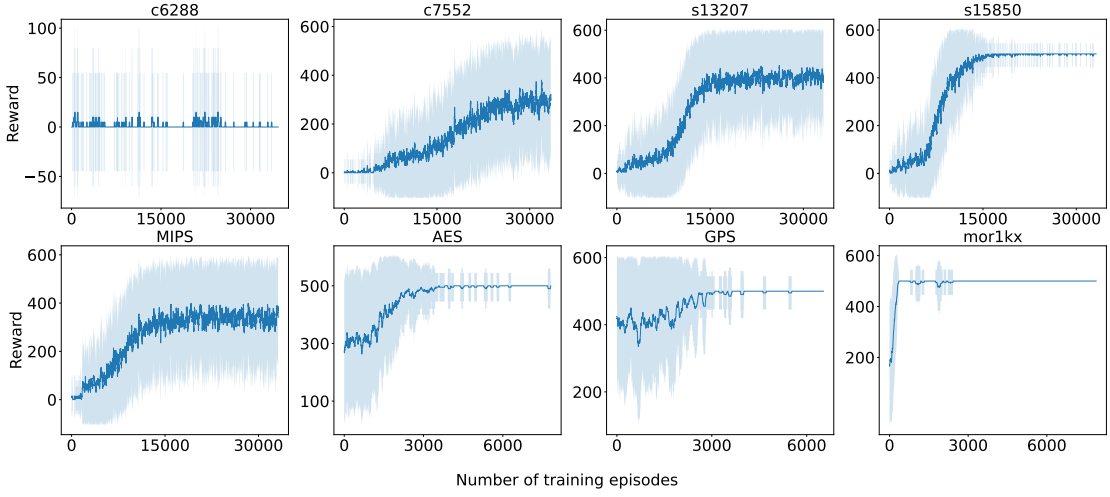

Figure 8: Rewards vs. number of training episodes. Shaded region represents the standard deviation of reward.

Table 12: Number of steps, episodes, and run time of training.

Design	c6288	c7552	s13207	s15850	MIPS	AES	GPS	mor1kx
Num. of Steps	100K	100K	100K	100K	100K	23.7K	19.9K	23.9K
Num. of Episodes	35.2K	33.6K	33.3K	33.3K	33.3K	7.9K	6.6K	7.9K
Training time (hrs.)	0.65	0.91	0.57	0.75	7.09	12	12	12

Table 13: Breakdown of attack success rate for HTs of different trigger widths generated by [59] against TARMAC [22].

Design	Attack Success Rate (%)				
	2-width HTs	3-width HTs	4-width HTs	5-width HTs	All HTs
c432	0	0	0	0	0
c880	0	0	0	0	0
c1355	0	0	0	0.1	0.02
c1908	0	0	0	0.6	0.15
c3540	0	0	0	0	0
c6288	0	0	0	0.25	0.02
Average	0	0	0	0.15	0.03

A.8 Analysis of ATTRITION Training Time

Here, we analyze the practicality of using ATTRITION in generating stealthy HTs. Table 12 depicts the number of steps, episodes, and the training time for each design. The solutions we developed (§4) enable ATTRITION to perform several thousand episodes ($> 23K$ on average) through which stealthy HTs are generated in less than 12 hours. In fact, for small designs like c6288, c7552, s13207, and s15850, the training ends in less than an hour. Medium-sized (MIPS) and large designs (AES, GPS, mor1kx) require about 7 to 12 hours of training to generate stealthy HTs. Even for the largest design (GPS with $> 190,000$ gates), the training time is bound by 12 hours. This training time is minimal compared to the three months allotted to foundries for fabrication [40]; hence, an adversary in the foundry has ample time to perform the attack. Finally, note that our developed solutions (§4) enable ATTRITION to perform several thousand episodes (> 23K on average) through which stealthy HTs are generated in less than 12 hours.

A.9 Waveforms for Case Studies

Figure 9 shows the simulation waveform for the privilege escalation attack on mor1kx. The mor1kx_cpu_cappuccino_mor1kx_cpu_mor1kx_ctrl_cappuccino_n_178 signal is the privilege bit of the processor, and the tmp_trig_en signal is the trigger signal of the HT. Initially, the processor starts in the supervisor mode (*i.e.,* privilege bit is 1) because the processor is designed in that way. Hence, to demonstrate a privilege escalation, we make the privilege bit 0 and then activate the HT. The HT is activated in the 13^{th} clock cycle, and immediately, the privilege bit escalates to 1, indicating a successful attack.

Similarly, Figure 10 shows the simulation waveform for the kill switch attack that prevents the processor from executing any useful instructions. The pc_fetch signal shows the 32–bit program counter, and the tmp_trig_en signal is the trigger that activates the HT. The HT is activated in the 19^{th} clock cycle, and impact of the HT is propagated to the target register, *i.e.,* the program counter, in the 21^{st} clock cycle. After that, the program counter gets stuck at 0x00000000, preventing further execution.

A.10 Reinforcement Learning for Attacks

Here, we discuss how RL has been used by researchers to develop attacks in the general domain of security. [75] proposed the first structural attack against graph-based Android malware detection techniques. [76] formulated a RL-guided framework that performs adversarial attacks in a black-box setting on state-of-the-art machine learning models for malware classification. [77] proposed a semantics-preserving RL-based attack against black-box graph neural networks for malware detection, achieving a significantly higher evasion rate than four state-of-the-art attacks. [78] showcased a RL-enabled malicious energy attack whereby judiciously choosing intelligent nodes in a green IoT network, an adversary can encourage the data traffic passing through a compromised node, thereby harming information security.

To the best of our knowledge, developing attacks using RL for hardware security problems has received little to no attention. In that light, ATTRITION showcases flawed security assumptions made by prior and state-of-the-art HT detection techniques across logic testing and side channel approaches by inserting stealthy HTs.

Table 14: Comparison of percentage of HTs with side-channel sensitivity less than 10%, for randomly generated and ATTRITION generated HTs against random, MERS, MERS-h, and MERS-s [24].

Technique \ Design		c6288	c7552	s13207	s15850	MIPS	AES	GPS	mor1kx
Random	Random HTs	100	100	100	100	100	100	100	100
	RL HTs	100	100	100	100	100	100	100	100
MERS [24]	Random HTs	100	100	100	100	100	100	100	100
	RL HTs	100	100	100	100	100	100	100	100
MERS-h [24]	Random HTs	83	100	100	100	100	100	100	100
	RL HTs	96	99	100	100	100	100	100	100
MERS-s [24]	Random HTs	80	100	100	100	100	100	100	100
	RL HTs	96	100	100	100	100	100	100	100

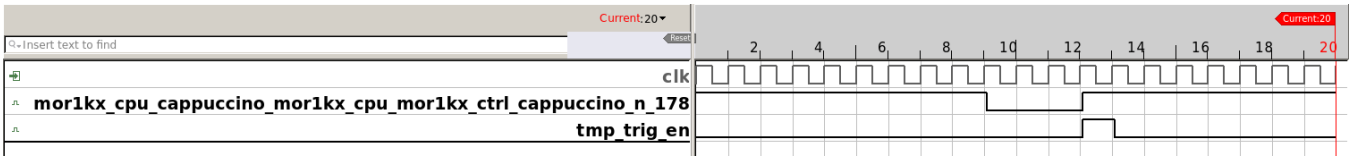

Figure 9: Simulation waveform for performing privilege escalation in mor1kx processor.

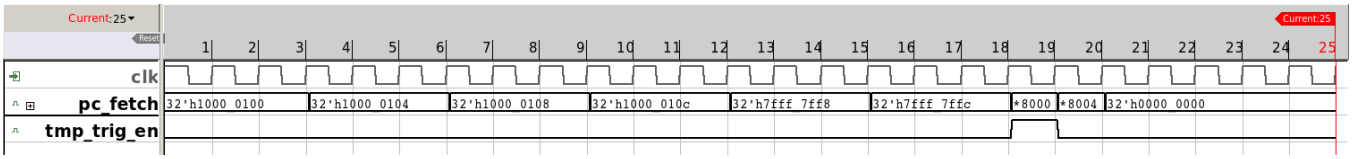

Figure 10: Simulation waveform for performing kill switch operation in mor1kx processor.

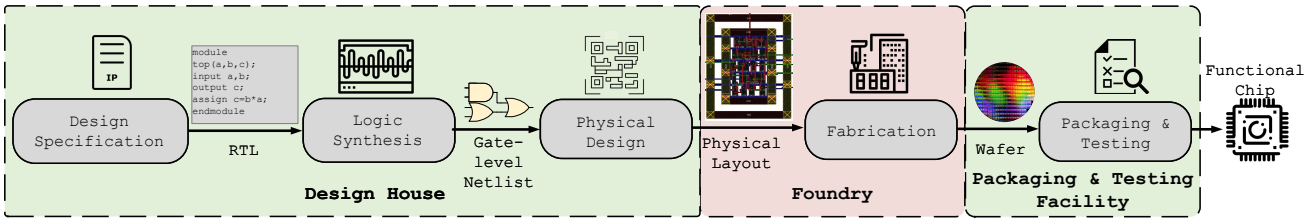

Figure 11: Integrated circuit design flow and supply chain. In our work, we assume a trusted design house and trusted packaging and testing facility. Fabrication facility (a.k.a. foundry) is untrusted.