2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)
Phoenix, Arizona, USA, August 6-9, 2023

Multi-criteria Hardware Trojan Detection: A Reinforcement Learning Approach

Amin Sarihi*, Peter Jamieson†, Ahmad Patooghy‡, Abdel-Hameed A. Badawy*
*Klipsch School of ECE, New Mexico State University, †Department of ECE, Miami University
‡Department of Computer Systems Technology, North Carolina A&T State University
*{sarihi, badawy}@nmsu.edu, †jamiespa@miamioh.edu, ‡apatooghy@ncat.edu

2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS) | 979-8-3503-0210-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/MWSCAS57524.2023.10406091

***Abstract*—Hardware Trojans (HTs) are undesired design or manufacturing modifications that can severely alter the security and functionality of digital integrated circuits. HTs can be inserted according to various design criteria, e.g., nets switching activity, observability, controllability, etc. However, to our knowledge, most HT detection methods are only based on a single criterion, i.e., nets switching activity. This paper proposes a multi-criteria reinforcement learning (RL) HT detection tool that features a tunable reward function for different HT detection scenarios. The tool allows for exploring existing detection strategies and can adapt new detection scenarios with minimal effort. We also propose a generic methodology for comparing HT detection methods fairly. Our preliminary results show an average of 84.2% successful HT detection in ISCAS-85 benchmarks.**

***Index Terms*—Reinforcement Learning, Hardware Trojan, Hardware Security.**

I. Introduction

Due to time-to-market constraints and increasing production costs, the integrated circuit (IC) supply chain has adopted a multi-party production model. According to this new model, most microelectronic chips are being produced outside of the country [1], raising security concerns about the design and fabrication of chips, particularly hardware Trojan (HT) insertion attacks.

Our current HT detection capabilities suffer from the following shortcomings. 1) Most detection methods perform the HT detection through a one-dimensional lens, i.e., nets' switching activity [2] and [3]. We believe that the current detection methods might not cover the real-world scenarios in which adversaries can insert HTs according to a range of criteria. 2) Available HT benchmarks suffer from significant limitations in size and variety of circuits, as well as the fact that they are all human-crafted and hence are biased by the expert mindset at the creation time [4][1].

This paper attempts to move the HT detection research space forward by developing a multi-criteria HT detector that explores many HT detection strategies, not limited to a designer's mindset. Our Reinforcement Learning (RL) HT detector has a tunable rewarding function that helps detect different HTs with different insertion strategies. The RL agent explores large circuit designs promptly and generates test vectors to find HTs in digital circuits. Our threat model consists of a security engineer that must verify a manufactured IC's integrity before allowing it to be integrated into a bigger design. The engineer only can rely on the golden netlist to produce test vectors. Our threat model inherently differs from previous works [2], where the design internals are still accessible in the pre-silicon phase. Our generated test patterns are publicly available through this link[2]. Additionally, this paper introduces a confidence value as a part of a methodology to compare HT detectors fairly. This helps security engineers to decide the merits of HT detectors for specific applications. In summary, the paper's contributions are as follows:

- We introduce an RL-based HT detection tool with a tunable rewarding function that can be modified and re-trained based on different criteria.
- We introduce and use a generic methodology to make fair comparisons among HT detectors.

The rest of the paper is organized as follows. We discuss previous endeavors in HT detection in Section II. Section III presents our HT detection tool. We define a security metric to better compare the HT detectors by security engineers in Section IV. Experimental evaluation of the proposed tool and analysis of the results are in Section V. Finally, Section VI concludes the paper.

II. Background and Previous Work

This section reviews existing hardware Trojan (HT) detection methods. MERO is a test pattern generator that tries to trigger possible HTs by exciting rare-active nets multiple times. MERO becomes less effective with larger circuits. Hasegawa et al. [5] extract 51 features from the Trusthub benchmarks and train a Random Forest classifier. However, the studied dataset is limited. Lyu et al. [2] proposed TARMAC, which maps the trigger activation problem to the clique

[1]The most referenced benchmarks are available on trust-hub.org.

[2]https://github.com/NMSU-PEARL/Hardware-Trojan-Insertion-and-Detection-with-Reinforcement-Learning

cover problem. TARMAC requires access to the internal nets and testing each suspect circuit separately. TGRL is an RL framework where the agent decides whether to flip a bit in the test vector according to an observed probability distribution. The reward function combines the number of activated nets and their SCOAP [6] (Sandia Controllability/Observability Analysis Program) parameters. The algorithm was not tested on any HT benchmarks and only test coverage was reported. DETERRENT [3] is another RL-based detector that finds the smallest set of test vectors to activate as many rare nets as possible; however, it only targets the switching activity of nets. HW2VEC [7] uses Graph Neural Networks to extract structural features from graphs and produce graph embeddings. The embeddings are passed to a deep neural network to classify circuits as HT-free or HT-infected. The detector is trained on Trusthub benchmarks. Unlike the previous work, our study proposes a multi-criteria RL-based HT detector tool that can detect HTs with different insertion strategies.

III. RL-BASED HT DETECTION

From an RL agent perspective for HT detection, the environment is a given circuit (or netlist) to determine whether it is clean or HT-infected. The agent interacts (performs an action) with the circuit by flipping input values to activate internal nets. The RL agent has an n-dimensional binary action space $a_t = [a_1, a_2, ..., a_n]$ where n is the number of circuit primary inputs. The agent may set or reset each a_i to transition to another state. $a_i = 0$ denotes that the value of the i^{th} input will remain unchanged from the previous test pattern, and $a_i = 1$ means that the input bit will flip. Attackers are likely to choose trigger nets with a consistent value (either 0 or 1) most of the time. Thus, a detector aims to activate as many dormant nets as possible. We consider two different approaches for identifying such rare nets:

1) Dynamic Simulation: We feed each circuit with $100K$ random test patterns and record the value of each net. Through logging nets transitions, we populate the switching activity statistics for each net and compare it against a threshold θ (ranges in $[0, 1]$). Nets with switching below θ are considered rare nets.

2) Controllability Simulation: This approach classifies the nets based on their *controllability*[3] values. Low switching nets have a high difference between their controllability value [8], *i.e.*, they are mostly stuck at 0 or 1. We set a threshold value η as defined in Eq. 1:

$$\eta = \frac{|CC1(Net_i) - CC0(Net_i)|}{Max(CC1(Net_i), CC0(Net_i))} \quad (1)$$

where $CC0(Net_i)$ and $CC1(Net_i)$ are the combinational controllability of 0 and 1 for Net_i, respectively. The η parameter ranges between $[0, 1)$ such that higher values of η correlate with lower net activity [8].

Our RL state is mapped to the set of the collected rare nets. In a circuit with m rare nets, the state space is defined as $State_t = [s_1, s_2, ..., s_m]$ where s_i is associated with the i^{th} net in the set. Whenever an action (a test pattern) activates s_i (taking its rare value), it will set that state to 1 in the state vector. Otherwise, its state stays at 0. As can be inferred, the action and state spaces are multi-binary. Figure 1 summarizes our tool flow.

A. Rewarding Functions

The agent's goal is to activate as many HT triggers as possible. Thus, a part of the rewarding function should enumerate rare nets. However, we should avoid over-counting situations where a rare net has successive dependent rare nets. We adopt a pruning strategy and pick the rarest net in a sequence of dependent rare nets (seen in Figure 1).

As for rewarding the agent, we consider three rewarding functions, and we explain them in the rest of this section. In our first rewarding function (hereafter $D1$), we use a copy of the agent's previous state and encourage it to generate states that differ from the previous one. This pushes the agent towards finding test vectors that lead to unseen states. The pruned current and previous state vectors are passed as inputs to $D1$; the final reward is the output. The reward function comprises an *immediate* and *sequential* parts. The sequential reward is computed by making a one-to-one comparison between the nets in the old and new states. The highest reward is given when an action can activate a net that was not triggered in the previous state, where it is given $+40$ for each net. If a rare net continues to be active in the new state, the agent will still be rewarded $+20$. The worst state transition is whenever an agent takes an action that leads to a rare net losing its rare value, and that is rewarded -3. Lastly, if the agent cannot activate a rare net after a state transition, it will be rewarded -1. The immediate award is the number of activated rare nets in the new state. Lastly, the final reward is a weighted mixture of immediate and sequential rewards with tunable weights. These parameters are selected such that the agent's collected rewards per episode would become more positive after initially starting with negative rewards.

Algorithm 1 describes our second rewarding function $D2$. In this case, the agent gains a reward proportional to the difficulty of the rare net it can trigger. This reward is computed using the inverse of net switching activities (line 4). If no vectors were found to trigger a net, it would be rewarded $10X$, the greatest reward in the vector (line 12). The algorithm encourages the agent to trigger the rarest nets in the circuit.

In the third rewarding function ($D3$), rare nets are populated based on threshold η in Eq. 1. When a rare net is activated,

[3]Controllability is the difficulty of setting a particular net to 0 or 1 value.

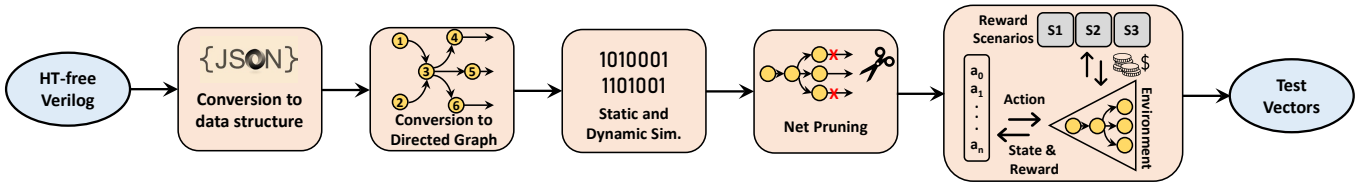

Fig. 1: The proposed toolset workflow.

Algorithm 1 Rewarding Function $D2$

Input: Net switching vector $Switching_{vector}$, Current state vector $State_{vector}$, State Vector Length K
Output: Final reward $Reward_{final}$

$Reward_{vector} = [0] * K$
for $k \in \{0, \ldots, K-1\}$ **do**
  if $(Switching_{vector}[k] != 0)$ **then**
    $Reward_{vector}[k] = Switching_{vector}[k]^{-1}$
  else
    $Reward_{vector}[k] = 0$
  end if
end for
$reward_{max} = max(Reward_{vector}[\,])$
for $k \in \{0, \ldots, K-1\}$ **do**
  if $(Switching_{vector}[k] == 0)$ **then**
    $Reward_{vector}[k] = 10 * reward_{max}$
  end if
end for
$Reward_{final} = 0$
for $k \in \{0, \ldots, K-1\}$ **do**
  if $(State_{vector}[k] == 1)$ **then**
    $Reward_{final} += Reward_{vector}[k]$
  else
    $Reward_{final} += -1$
  end if
end for

the agent is rewarded with the controllability of the rare value. This scenario aims to investigate controllability-based HT detection using an RL algorithm.

IV. The Proposed Generic HT-Detection Metric

We propose the following methodology to the community for fair and repeatable comparisons among HT detection methods. This methodology obtains a confidence value that one can use to conduct a fair comparison between different HT detection methods. There are 4 possible outcomes when an HT detection tool studies a given circuit. From the tool user's point of view, the outcomes are probabilistic events. For example, when an HT-free circuit is being tested, the detecting tool may either classify it as an infected or a clean circuit, *i.e.*, $Prob(FP) + Prob(TN) = 1$ where FP and TN stand for *False Positive* and *True Negative* events. Similarly, for HT-infected circuits, we have $Prob(FN) + Prob(TP) = 1$. We know FN and FP are two undesirable outcomes that detectors misclassify. Between these two, FN cases are much more dangerous because an FN case leads to a situation in which we rely on an HT-infected chip, whereas an FP case means wasting a clean chip by either not selling or not using it. So, we need to know how HT detection tools' user (might be a security engineer or a company representative) prioritizes FN and FP cases. We define a parameter α as the ratio of the undesirability of FN over FP. The tool user determines α based on characteristics and details of the application that eventual chips will be employed in, *e.g.*, the risks of using an infected chip in a device with a sensitive application versus using a chip for home appliances. Note that the user sets this value, which is not derived from the actual FP and FN. After α is set, it is plugged in Eq. 2 and a general confidence basis $Conf.Val$ is computed.

$$Conf.Val = \frac{(1 - FP)}{(1/\alpha + FN)} \tag{2}$$

This metric can make a fair comparison between HT detection methods regardless of their detection criteria and implementation methodology. The defined confidence metric combines the two undesirable cases concerning their severity from the security engineer's point of view, and it ranges between $[\frac{0.5\alpha}{1+0.5\alpha}..\alpha]$. The closer the value is to α, the higher the confidence in the detector. The absolute minimum of the $Conf.Val = 1/3$ that happens when $\alpha = 1$ and $FP = FN = 50\%$. This analysis assumes that FN and FP are independent probabilities. We note that for some detection methods, FP is always 0. For instance, test-based HT detection methods that use a golden model (HT-free) circuit for comparison and decision-making. However, our metric is general and captures such cases.

V. Experimental Evaluations

Our proposed multi-criteria HT detector is developed in Python. The training process of the RL agent is done using the PPO (proximal policy optimization) [9] from the Stable Baselines library with an episode length of 10. This guarantees that the agent would reset each 10 episodes and agent observes

TABLE I: Detection accuracy of scenarios D1, D2, and D3 for HTs with different input widths in [4].

Benchmark	2-Input HT			3-Input HT			4-Input HT			5-Input HT		
	D1	D2	D3	D1	D2	D3	D1	D2	D3	D1	D2	D3
c432	15.0%	27.8%	8.0%	41.2%	61.9%	42.8%	36.4%	66.5%	24.9%	24%	49.1%	21.6%
c880	100%	100%	100%	100%	92.0%	84.0%	86.7%	83.0%	64.1%	85.1%	79.3%	40.2%
c1355	94.6%	99.1%	98.3%	92%	98.1%	97.5%	90.5%	97.7%	96.2%	89.6%	97.0%	95.5%
c1908	96.4%	98.3%	97.3%	97.6%	96.4%	94.9%	93.0%	93.2%	94.4%	89.6%	91.1%	86.7%
c3540	56.0%	89.8%	89.8%	86.5%	91.6%	91.0%	89.4%	97.1%	98.8%	77.8%	79.2%	83.9%
c6288	96.1%	97.9%	97.1%	97.5%	97.6%	97.2%	95.6%	96.1%	95.9%	93.3%	94.1%	93.7%
Confidence Value	**2.97**	**4.07**	**3.19**	**4.13**	**4.89**	**3.93**	**3.56**	**4.73**	**3.23**	**2.91**	**3.52**	**2.51**

a new state. We select six circuits from ISCAS-85, namely $c432$, $c880$, $c1355$, $c1908$, $c3540$, and $c6288$.

To accelerate the agent's training time, instead of calling time-consuming graph functions, we built adjacency matrices and dictionaries that contain structural information of each node within the graph. This efficient technique speeds up training and testing processes by $3.7\times$ and $3.2\times$, respectively.

We start from $450K$ of timesteps for training in $c432$ and increase the timesteps for each successive circuit by 10% to enable enough exploration for larger circuits. We ran the training processes in parallel for each circuit. This process took nearly 27 hours to train the benchmark set. In the testing phase, we ran the trained RL agent for $20K$ episodes. To select a test vector, we set a cut-off reward of one-tenth of the collected reward in the last training episode (since we have ten steps per episode). We gather $20K$ test vectors that surpass this reward threshold.

Table I summarizes the detection percentages of our three detection scenarios for different HT sizes inserted in ISCAS-85 [4]. The inserted HTs in this dataset were introduced to address two issues: 1) removing inherent human bias in current HT databases and 2) providing ample HT instances for training detectors. Table I lists six benchmarks with HTs triggered by 2, 3, 4, and 5 input wires and reports the detection accuracy for $D1$, $D2$, and $D3$ (labeled across the top of the table). The number of HTs for each case is in [4].

From the table, $D2$ has the best detection rate in most cases; however, exceptions exist. For instance, in $c880$, the detection rate for $D1$ is equal to or better than $D2$, especially for 5-input HTs. The same happens for 3-input HTS in $c1908$.

On the other hand, $D3$ shows its superiority in $c3540$. Except for the 3-input Trojans, $D3$ equals or is better than the other two rewarding scenarios. This underlines the importance of $D3$, which uses an inherently different detection criterion. Polling among the three HT detection scenarios can generally lead to satisfactory HT detection in most circuits.

One interesting observation concerns the detection rate of $c432$. While applying $100,000$ random test vectors, we found that the rarest net in the circuit was triggered 7% of the time, which is significantly higher than other circuits where many nets exhibit switching activity of less than 1%. This suggests that the inserted HTs in the $c432$ might be activated easier with random test vectors. To test this hypothesis, we generated $20,000$ additional random test vectors and applied them to the circuit, detecting 99% of the HTs. This demonstrates that the RL attack did not have the intended impact in $c432$.

The confidence metric of our HT-detection tool proposed in Section IV can be seen in the last row of the table, assuming $\alpha = 10$. The confidence value for each column is calculated by averaging each detection scenario for each column. The table shows that the security engineer can put confidence in the $D2$ detector since it has higher confidence values than the other detection scenarios. The confidence value only surpasses 5 for detectors with $FN < 10\%$. In other words, the confidence value increases sharply, and better HT detectors are rewarded with more exponential confidence values.

VI. Conclusions

This paper emphasizes the need for multi-criteria HT detection tools and universal metrics to compare them. We propose a reinforcement learning tool for hardware Trojan detection, which features three rewarding functions that detect a wide range of HTs. Results on ISCAS-85 circuits showed a high detection rate of the proposed tool for various HTs. We also present a methodology to help the community compare HT detection methods regardless of their implementation details. We applied the methodology to our HT detection and discovered that our tool offers the highest confidence in HT detection when using the rewarding function D2.

References

[1] "Securing defense-critical supply chains: An action plan developed in response to president biden's executive order 14017."

[2] Y. Lyu and P. 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*, vol. 40, no. 7, pp. 1287–1300, 2020.

[3] V. Gohil, S. Patnaik, H. Guo, D. Kalathil, and J. Rajendran, "Deterrent: detecting trojans using reinforcement learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 697–702.

[4] A. Sarihi, A. Patooghy, P. Jamieson, and A.-H. A. Badawy, "Hardware trojan insertion using reinforcement learning," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, 2022, pp. 139–142.

[5] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[6] L. Goldstein and E. Thigpen, "Scoap: Sandia controllability/observability analysis program," in *17th Design Automation Conference*, 1980, pp. 190–196.

[7] S.-Y. Yu, R. Yasaei, Q. Zhou, T. Nguyen, and M. A. Al Faruque, "Hw2vec: A graph learning tool for automating hardware security," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 13–23.

[8] S. M. Sebt, A. Patooghy, H. Beitollahi, and M. Kinsy, "Circuit enclaves susceptible to hardware trojans insertion at gate-level designs," *IET Computers & Digital Techniques*, vol. 12, no. 6, pp. 251–257, 2018.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.