Hardware Trojan Insertion Using Reinforcement Learning

Amin Sarihi
sarihi@nmsu.edu
New Mexico State University

Ahmad Patooghy
North Carolina A&T State University

Peter Jamieson
Miami University

Abdel-Hameed A. Badawy
New Mexico State University

ABSTRACT
This paper utilizes Reinforcement Learning (RL) as a means to automate the Hardware Trojan (HT) insertion process to eliminate the inherent human biases that limit the development of robust HT detection methods. An RL agent explores the design space and finds circuit locations that are best for keeping inserted HTs hidden. To achieve this, a digital circuit is converted to an environment in which an RL agent inserts HTs such that the cumulative reward is maximized. Our toolset can insert combinational HTs into the ISCAS-85 benchmark suite with variations in HT size and triggering conditions. Experimental results show that the toolset achieves high input coverage rates (100% in two benchmark circuits) that confirms its effectiveness. Also, the inserted HTs have shown a minimal footprint and rare activation probability.

CCS CONCEPTS
• Security and privacy → Embedded systems security; Malicious design modifications; Malicious design modifications.

KEYWORDS
Hardware Trojan, Reinforcement learning, Automated Benchmarks

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Amin Sarihi, Ahmad Patooghy, Peter Jamieson, and Abdel-Hameed A. Badawy. 2022. Hardware Trojan Insertion Using Reinforcement Learning. In Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI ’22), June 6–8, 2022, Irvine, CA, USA. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3526241.3530379

1 INTRODUCTION
With recent advancements in VLSI technologies, integrated circuits (ICs) manufacturing involves multiple companies, introducing new security challenges at each stage of IC production. Hardware Trojans (HTs) are defined as any undesired modification in an IC which can lead to erroneous outputs or leak of information [7]. According to the adversarial model introduced by Shakya et. al. [15], HTs can be inserted into the target IC in three different scenarios, namely intellectual properties (IPs) (processing cores, various I/O components, and network-on-chip [12]), disgruntled employees at the integration stage, reverse-engineering by an untrusted foundry. To study the behavior of HTs in digital circuits, researchers have been mostly using limited benchmarks [11, 15] including a set of 91 trust benchmarks with different HT sizes and configurations (available at trust-hub.org). Over the past years, various HT detection approaches have been developed based on these benchmarks [6, 9, 10, 14]. Despite the valuable effort to create such HT benchmarks, they are limited in size and variety necessary to push the detection tools. Another downside of using these existing benchmarks is the problem of dealing with fixed static trigger conditions [3], which stems from the human bias during HT insertions. As a result, HT detectors can be tuned in a way to enhance their detection accuracy while not truly being effective in complex and real-world modern ICs. These shortcomings emphasize the need for an automated HT-insertion tool free of human biases that can be used to create high-volume HT benchmarks. Such a tool will help implement HT benchmarks aligned with the fast growth of attack approaches and cater to the security needs in the realm of hardware design. By introducing new HTs into a design, one can create new benchmarks and push the capabilities of HT detection via introducing never-seen-before HTs.

Although a few researchers have tried to address these problems by introducing tunable HT insertion toolsets [3, 18], these approaches have no concrete guideline for selecting trigger and payload nets; instead, triggering is done on an ad-hoc basis with little design space exploration capabilities. In this paper, we attempt to develop an HT-insertion tool, free of human biases, using a Reinforcement Learning (RL) agent that decides where to insert HTs through a trial and error method. Although machine learning techniques have been used to detect HTs in the past [6, 10, 17], to the best of our knowledge, this work is the first that addresses HT insertion using a machine learning approach via design space exploration.

Our toolset translates each circuit to a graph representation in which different properties of each net, such as controllability, observability, and logical depth, are computed (so-called the SCOAP* parameters [4]). The circuit graph is considered as an environment in which the RL agent tries to insert the HT to maximize the gained rewards. Obtained results confirm that the inserted HTs are very hard to detect as the toolset maximizes the number of IC’s inputs involved in the activation of the inserted HTs. We define a metric called the input coverage percentage (ICP) to determine the difficulty of HT activation.

*Sandia controllability and observability analysis program.
The paper is organized as follows: The mechanics of our proposed approach are presented in Section 2. Section 3 demonstrates the experimental results and Section 4 concludes the paper.

## 2 RL-BASED HT INSERTION

Figure 1 shows the flow of the proposed HT insertion tool. The first step to insert an HT into a circuit is to create a graph representation of the flattened netlist from the circuit. Yosys Open Synthesis Suite [16] translates a Verilog file of the circuit into a JSON (JavaScript Object Notation) [1] netlist where the JSON file is used by a python script to parse the internal graph representation of the circuit. Next, the tool finds a set of rare nets to be used as HT trigger nets (this is described in detail in Subsection 2.1). Finally, an RL agent uses the rare net information and attempts to insert an HT to maximize a reward function as described in Subsection 2.2.

### 2.1 Rare Nets Extraction

As discussed earlier, different circuit criteria have been used for trigger selection. In this work, we use the parameters introduced in [14] where the trigger nets are selected based on functions of net controllability and observability [4].

The first parameter is called the HT trigger susceptibility parameter, and it is derived from the fact that low switching nets have a high difference between their controllability values. Equation 1 describes this parameter:

\[
HTS(Net_i) = \frac{|CC1(Net_i) - CC0(Net_i)|}{\text{Max}(CC1(Net_i), CC0(Net_i))} \tag{1}
\]

where HTS is the HT trigger susceptibility parameter of a net; CC0(Neti) and CC1(Neti) are the combinational controllability 0 and 1 of Neti, respectively. The HTS parameter ranges between [0, 1) such that higher values of HTS are correlated with lower activity on the net.

The other used parameter is specified in Equation 2 to measure the ratio of observability to controllability:

\[
OCR(Net_i) = \frac{CO(Net_i)}{CC1(Net_i) + CC0(Net_i)} \tag{2}
\]

where OCR is the observability to controllability ratio of a net. This equation requires that the HT trigger nets must be very hard to control but not so hard to observe. Unlike the HTS parameter, OCR is not bounded, and it belongs to the interval of [0, 1]. We will specify thresholds (see Section 2.2) for each parameter and use them as filters to populate the set of rarely-activated nets for our tool.

### 2.2 RL-Based HT Insertion

Agent, Action, Environment, State, and Reward are the five main components of reinforcement learning. From an RL perspective, we define the environment as the circuit in which we are trying to insert HTs. The agent’s action is the insertion of the HT. Note, we consider combinational HTs where trigger nets are ANDED, and the payload is an XOR gate.

We represent different HT insertions with a state vector in each given HT, the state vector is comprised of

\[
\begin{align*}
\text{level} & = \text{One of the five levels of an circuit. To address this issue, we first levelize the circuit. The output level of an } m \text{-input gate is computed by equation } 3: \\
\text{Level(out)} &= \text{MAX}(	ext{Level(in1)}, \text{Level(in2)}, \ldots, \text{Level(in}_m)) + 1 \tag{3}
\end{align*}
\]

Figure 2 depicts a 2-input HT (in yellow) where the XOR payload flips the value of the target net when the trigger is activated. For a given HT, the state vector is comprised of

\[
s = [s_1, s_2, \ldots, s_{n-2}, s_{n-1}, s_n]
\]

where s1 through s_{n-2} are levels of the HT inputs and s_{n-1} and s_n are levels of the target net and the output of the XOR payload, respectively. As an example, the HT in Figure 2 has the state vector \(s = [2, 1, 3, 4]\). The action space of the described HT agent is multi-discrete, i.e., each input of the HT can choose an action from a set of 5 available actions. These actions are:

- **Next level**: the input of the HT moves to one of the nets that is one level higher than the current net level.
- **Previous level**: the input of the HT moves to one of the nets that is one level lower than the current net level.
- **Same level up**: the input of the HT will move to one of the nets at the same level as the current net level. The net is picked by pointing to the previous net in the ascending list of net ids for the given level.
- **Same level down**: the input of the HT will move to one of the nets at the same level as the current net level. The net is picked by pointing to the previous net in the ascending list of nets for the given level.
- **No action**: the input of the HT will not move.

If an action leads the agent to step outside the circuit boundaries, it is substituted with a "No action".

The action space is also represented by a vector where its size is equal to the number of the HT inputs, and each action can be one of the five actions above, e.g., for the HT in Figure 2, the action space would be \(a = [a_1, a_2]\) since it has two inputs.

As we explained in Section 2.1, SCOAP parameters are first computed. We specify two thresholds \(T_{HTS}\) and \(T_{OCR}\) and require our algorithm to find nets that have higher HTS values than \(T_{HTS}\) and lower OCR values than \(T_{OCR}\). These nets are classified as suspicious nets.

Our toolset utilizes an algorithm that consists of two conditional while loops that keep track of the terminal states and the elapsed timesteps. One important function used in the algorithm is called \(reset\text{-environment}\) which resets the environment before each episode. Upon reset, an HT is randomly inserted within the circuit according to the following set of rules:

- Rule 1) Trigger nets are selected randomly from the list of the total nets.
- Rule 2) No trigger net is allowed to be fed from a previously used net.
- Rule 3) Trigger nets cannot be assigned as the target.
- Rule 4) The target net is selected considering the level of trigger nets. To prevent forming combinational loops, we specify that the level of the target net should be greater than that of the trigger nets.

During the training process in each episode, we do not change the payload net to help the RL algorithm converge faster for possible solutions. Unlike the manual payload selection, we allow the algorithm to explore the environment during different episodes and decide which payload can more seriously compromise the security by collecting higher rewards. This solution addresses the problem of finding optimal payload selection.
When the trigger nets do not belong to the set of suspicious nets, the toolset outputs Verilog gate-level netlist files that contain the malicious HTs.

Our proposed RL rewarding scheme drives the agent towards the vicinity of the suspicious nets (states) where rewards are more positive and hence result in stealthier HTs. The agent is rewarded when the HT can be triggered in the objective function \[13\]. At last, the algorithm more time to explore the environment and keeps the experimental setup relatively consistent as the HT size increases. The training of the RL agent is done using Stable Baselines library \[8\] with MLP (multi-layer perceptron) as the PPO algorithm policy. For testing, the agent is managed to insert HTs into the benchmarks, which are converted into equivalent circuit graph using NetworkX \[5\].

In our experiments, we have specified \( T_{OCP} \) and \( T_{HTS} \) such that 5% of all nets in each circuit are considered as suspicious nets. This was done to enable a fair comparison between the circuits. Also, we increase the number of total timesteps and steps per episode by 10% every time we increase the HT input size by one. This gives the algorithm more time to explore the environment and keeps the experimental setup relatively consistent as the HT size increases. Our initial total timestep for each circuit is 120000 steps for an HT of size two.
Table 1: Number of inserted HTs with different size and percentage of input engagement

| Benchmark | 2-Input HT | 3-Input HT | 4-Input HT | 5-Input HT |
|-----------|------------|------------|------------|------------|
|           | > 60% | > 63% | > 70% | > 60% | > 65% | > 70% | > 60% | > 65% | > 70% |
| c432      | 80    | 47    | 10    | 23    | 160   | 23    | 477   | 167   | 30    |
| c880      | 1     | 0     | 0     | 0     | 47    | 0     | 170   | 15    | 1     |
| c1355     | 9390  | 13180 | 10749 | 9127  | 19407 | 16619 | 13702 | 25211 | 22510 |
| c1908     | 8227  | 22670 | 15446 | 9543  | 17792 | 13559 | 8263  | 32820 | 26063 |
| c3540     | 161   | 140   | 50    | 6     | 280   | 64    | 208   | 687   | 313   |
| c6288     | 9967  | 16113 | 15951 | 15119 | 24178 | 22312 | 20974 | 32111 | 30954 |

Table 1 shows the number of inserted HTs under various ICPs greater than thresholds (60%, 65%, and 70%) in six circuit benchmarks for different HT sizes. As can be observed in most cases, as the number of HT inputs increases, our tool can insert more HTs with higher ICP into the circuit. This stems from the fact that more HT inputs broaden the possible logic cones that can be activated. Subsequently, more inputs are engaged in the HT activation process. For example, the HT input size increases, the toolset is able to insert HTs with higher ICP in c432, c880, c1908, and c3540. The toolset is able to insert HTs in c1355 and c6288 with an ICP of 100% even with 2-input HT and just below 100% in c1908.

Figure 4 depicts the number of inserted HTs against the ICP in c432. The agent was able to insert a 5-input HT that covers 97% of the inputs, while the smaller HTs were unable to achieve such ICP; however, this high ICP comes with the cost of a bigger HT in the circuit that could disclose its presence. Interestingly, the 2-input HT has achieved 81% ICP. Hence the attacker can strike a balance between the HT footprint and its trigger coverage.

The impact of inherent randomness in choosing an Agent action or resetting the environment in the RL algorithm is observed in the results for c3540, where the number of found inserted 3-inputs HTs is less than the 2-input HTs. The results partly rely on what state the agent starts from and what sequence of actions the agent takes.

Another observation is how different benchmark circuits differ in the overall magnitude of inserted HTs. Our hypothesis is that the functionality of a circuit impacts how easy it is to insert HTs. As a case study, we investigate the different characteristics of circuits c880 and c1355. We first compute the average input access of all the target nets in the inserted HTs set, which indicates a path between all the inputs and the target nets. This number is 53% in c880 while it is 88% in c1355. Hence, the target nets can access more inputs on average in c1355. Moreover, an analysis of the suspicious nets in both circuits indicates that, unlike c880, there are different sets of nets in c1355 that share the same HTs and OCR parameters, which we call buses. There are two 32-bit buses in the suspicious nets set. We observed that the agent only needs one trigger net from these two buses to provide high ICP. Hence, the number of possible inserted HTs requiring only one such net is higher compared to c880.

4 CONCLUSION

This paper presented an automated RL HT insertion toolset to address the problem of human bias in HT benchmark design. The rewarding scheme in our study was defined such that the agent finds the HTs that can engage as many inputs as possible for HT activation. We achieved this by training an RL agent, and unlike the previous work, where normal trigger nets were chosen randomly, our agent automatically picked a set of normal nets alongside the rare nets to maximize the cumulative rewards of the learning agent. This new method of inserting HTs in the ISCAS-85 benchmark circuits can help researchers in the community to devise better HT detection techniques by increasing the HT benchmark set.

REFERENCES

[1] L. Bassett. 2015. Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON. O’Reilly Media. https://books.google.com/books?id=Qv9PCgAAQBAJ
[2] Michael L. Bushnell. 2000. Essentials of electronic testing for digital. Memory & Mixed-Signal VLSI Circuits (2000)
[3] Jonathan Cruz, Yuanwen Huang, Prabhat Mishra, and Swapar Bhunia. 2018. An automated configurable Trojan insertion framework for dynamic trust benchmarks. In 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 1598–1603.
[4] Lawrence H Goldstein and Evelyn L Thigpen. 1980. SCOAP: Sandia controllability/observability analysis program. In Proceedings of the 17th Design Automation Conference. 190–196.
[5] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure. Dynamics, and Function using NetworkX. In Proceedings of the 7th Python in Science Conference, Gael Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.
[6] Kento Hasegawa, Masao Tanagawa, and Nozomu Togawa. 2017. 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, 1–4.
[7] Zhaspar Pan and Prabhat Mishra. 2021. Automated test generation for hardware trojan detection using reinforcement learning. In Proceedings of the 26th Asia and South Pacific Design Automation Conference. 408–413.
[8] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Ankit Kanervisto, and Noah Dorrnann. 2019. Stable baselines3. GitHub repository (2019).
[9] Mohamad Sabri, Ahmad Shabani, and Bijan Alizadeh. 2021. SAT-Based Integrated Hardware Trojan Detection and Localization Approach Through Path-Delay Analysis. IEEE Transactions on Circuits and Systems II: Express Briefs (2021).
[10] Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. 2013. On design vulnerability analysis and trust benchmarks development. In 2013 IEEE 31st International Conference on Computer Design (ICCD). IEEE, 471–474.
[11] Amin Sarabi, Ahmad Pataghoy, Ahmed Khalid, Mahdi Hasanazadeh, Mostafa Said, and Abdel-Hamed A Badway. 2021. A Survey on the Security of Wired, Wireless, and 3D Network-on-Chips. IEEE Access (2021).
[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707:06347 (2017)
[13] Seyed Mohammad Sebt, Ahmad Patooghy, Ahmed Khalid, Mahdi Hasanzadeh, Mostafa Said, and Abdel-Hamed A Badway. 2021. A Survey on the Security of Wired, Wireless, and 3D Network-on-Chips. IEEE Access (2021).
[14] Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707:06347 (2017)
[15] Mingfu Xue, Chongyang Gu, Weiqiang Liu, Shichao Yu, and Márice O’Neill. 2020. Ten years of hardware Trojans: a survey from the attacker’s perspective. IET Computers & Digital Techniques 12, 6 (2018), 251–257.
[16] Bicky Shyaka, Tony He, Hassan Salami, Domenic Forte, Swapar Bhunia, and Mark Tehranipoor. 2017. Benchmarking of hardware trojans and maliciously affected circuits. Journal of Hardware and Systems Security 1, 1 (2017), 85–102.
[17] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys—a free Verilog synthesis suite. In Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip).