An Automated Configurable Trojan Insertion Framework for Dynamic Trust Benchmarks

Jonathan Cruz[1], Yuanwen Huang[2], Prabhat Mishra[2], and Swarup Bhunia[1]
[1] Department of Electrical & Computer Engineering
[2] Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL, USA

***Abstract*—Malicious hardware modification, also known as hardware Trojan attack, has emerged as a serious security concern for electronic systems. Such attacks compromise the basic premise of hardware root of trust. Over the past decade, significant research efforts have been directed to carefully analyze the trust issues arising from hardware Trojans and to protect against them. This vast body of work often needs to rely on well-defined set of trust benchmarks that can reliably evaluate the effectiveness of the protection methods. In recent past, efforts have been made to develop a benchmark suite to analyze the effectiveness of pre-silicon Trojan detection and prevention methodologies. However, there are only a limited number of Trojan inserted benchmarks available. Moreover, there is an inherent bias as the researcher is aware of Trojan properties such as location and trigger condition since the current benchmarks are static. In order to create an unbiased and robust benchmark suite to evaluate the effectiveness of any protection technique, we have developed a comprehensive framework of automatic hardware Trojan insertion. Given a netlist, the framework will automatically generate a design with single or multiple Trojan instances based user-specified Trojan properties. It allows a wide variety of configurations, such as the type of Trojan, Trojan activation probability, number of triggers, and choice of payload. The tool ensures that the inserted Trojan is a valid one and allow for provisions to optimize the Trojan footprint (area and switching). Experiments demonstrate that a state-of-the-art Trojan detection technique provides poor efficacy when using benchmarks generated by our tool. This tool is available for download from http://www.trust-hub.org/.**

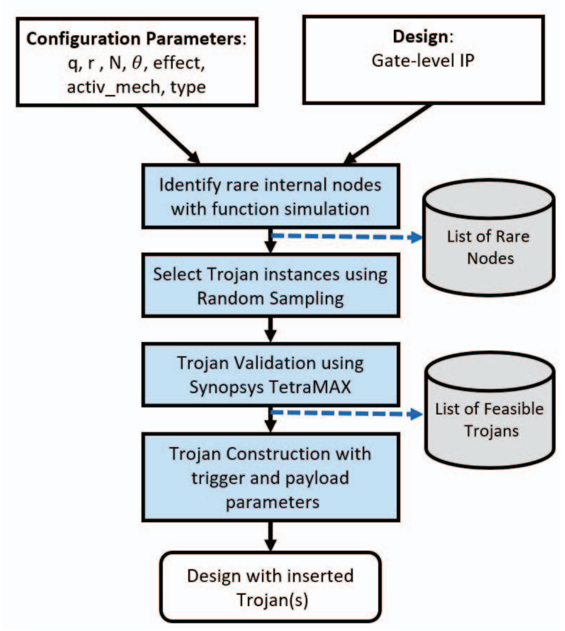

Configuration Parameters:
q, r , N, θ, effect, activ_mech, type
Design:
Gate-level IP
Identify rare internal nodes with function simulation
List of Rare Nodes
Select Trojan instances using Random Sampling
Trojan Validation using Synopsys TetraMAX
List of Feasible Trojans
Trojan Construction with trigger and payload parameters
Design with inserted Trojan(s)

Fig. 1: Tool flow and integration with commercial test tool in the proposed Trojan insertion framework.

I. Introduction

With research in hardware security and trust increasing in recent years, benchmarks serve as an important tool for researchers to assess the effectiveness of their proposed methodologies by providing a standardized baseline. Traditionally, Trojans have been inserted in an ad-hoc manner into pre-silicon designs. This fact prevents effective comparison among Trojan detection methods because an ad-hoc Trojan can favor one detection methodology over another. Recent efforts have attempted to remedy this by offering a benchmark suite with fixed number of Trojan inserted designs [10], [11]. However, existing trust benchmarks have the following major deficiencies: (1) These benchmarks only enumerate a subset of the possible hardware Trojans. There exists an inherent bias in these designs as the Trojan location and trigger conditions are static. As a result, it is possible for researchers to tune their methods (often unknowingly) to detect these Trojans. (2) A static set of benchmarks also prevent us from incorporating new types of Trojans in this rapidly evolving field, which keep discovering new Trojan structures. (3) A limited number of benchmarks can negatively affect supervised machine learning techniques for Trojan detection which require an expansive test set. (4) Finally, the existing set does not allow inserting multiple Trojans in a design or inserting Trojan an a different design, e.g. a new intellectual property (IP) block.

In order to make a more robust and flexible Trust benchmark suite, we have developed a novel framework to dynamically insert various functional Trojan types into a gate-level design. The possible Trojans that can be inserted using our tool can vary in terms of the Trojan type (e.g., combinational, sequential, etc.), the number of Trojans, detection difficulty, number and rarity of trigger points, payload types, and Trojan structure. For added flexibility and forward-compatibility, we also allow users to insert a *Template Trojan* with a chosen payload and triggering condition from a combination of rare or non-rare nodes. Figure 1 provides an overview of our proposed framework. We first identify the rare internal nodes in the given netlist. Potential Trojan instances are generated using the principle of random sampling from the population

This work was partially supported by grants from NSF (CNS-1441667, 1603483, 1623310), SRC (2014-TS-2554, 2651) and Cisco.

of rare nodes (and non-rare if specified). Trigger conditions and payloads are verified producing a feasible Trojan list. From this list, we then randomly select and insert the Trojans according to the user's configuration options including any footprint optimization. Our Trojan insertion framework is extensively tested and ready to be released on TrustHUB web portal (http://www.trust-hub.org) for use by the broader research community. We also use our benchmarks to evaluate a well-known Trojan detection technique, COTD [13], and show that it can fail to detect Trojans generated by our tool while being effective to detect Trojans from static benchmarks [10], [11].

To the best of our knowledge, this is the first paper that describes a tool flow for inserting custom Trojans with validated payload and trigger conditions in gate-level designs. Our specific contributions are as follows:

1) An algorithm is developed and used for analysis and validation of potential triggers and Trojan payloads. The algorithm interfaces with commercial test tools to ensure validity of random Trojans.
2) The framework provides a highly flexible and customizable interface for Trojan insertion based on user-specified Trojan properties. It allows insertion of a single Trojan instance with functional change or leakage effects.
3) It is extended to allow for multiple Trojan insertions of different types on a single gate-level design. Further, it makes provision for optimizing Trojan footprint to make it difficult with respect to side-channel analysis based Trojan detection [3].

The remainder of the paper is as follows. Section 2 discusses previous benchmark development efforts. Section 3 describes the background and threat model. Section 4 presents the tool flow and configurations. Section 5 presents evaluation results using a state-of-the-art Trojan detection method. Finally, Section 6 concludes the paper.

II. Related Work

Benchmarks provide a set of standard designs for assessing different Trojan detection and prevention methods in the hardware security community. During the earlier years of nearly a decade of hardware Trojan research, due to lack of well-specified benchmarks, researchers inserted Trojans in an ad-hoc manner in various designs that no longer allows for a fair comparison of different Trojan detection and prevention techniques [1], [2], [5], [6]. Recent development of a set of 91 Trust-Hub [10], [11] benchmarks that come with hard-to-activate Trojans has been a valuable contribution in this field. These benchmarks contain Trojans of varying size and attack models offering researchers an opportunity to test their methods. This prominent effort in standardization for hardware trust evaluation is a step in the right direction, but there are specific limitations as described in Section I that need to be addressed.

Our proposed approach addresses these limitations by enabling the generation of a wide variety of unbiased benchmarks and offers the following benefits over a static benchmark suite:

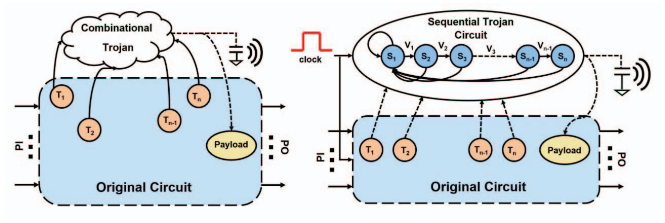


Fig. 2: (a) Generic Combinational Trojan; (b) Generic Sequential Trojan.

- It creates a more representative model of entire Trojan population through random and stratified sampling.
- It allows for an automated red vs blue team scenario when evaluating detection methods.
- It facilitates incorporation of new Trojan attack models and structures.

III. Background and Motivation

A. Trojan Model

Hardware Trojans are malicious modifications of a circuit that cause undesired side-effects. Trojans structurally consist of a trigger and a payload. Typically, an attacker will insert Trojans under hard-to-trigger conditions or rare nodes of a design. Therefore, most Trojans lay dormant for a majority of an infected circuit's life-time and subsequently evade detection when using standard validation techniques. Once the trigger or activation condition is reached, the Trojan's effect is realized through the payload gate. These payload effects can be broadly classified as functional *denial of service* (DoS) or *information leakage* [1], [2].

There are two general structures of functional Trojans: combinational and sequential. Combinational Trojans consist of only combinational logic gates. Figure 2 (a) shows a generic combinational Trojan. Simultaneous activation of rare nodes T_1 to T_n leads to undesired effects. Sequential Trojans include state elements and a series of state transitions to trigger an undesired effect. In Figure 2(b), after the state transitions from S_1 to S_n occur, the Trojan is activated. These state transitions can be activated from trigger nodes T_1 to T_n or from just the clock signal in the case of always-on Trojans.

B. Trojan Detection

Trojan detection techniques focus on identifying Trojans that are inserted by malicious adversaries along the supply chain. We divide the approaches into three broad categories:

Functional Testing based Trojan detection methods involve generating test sets aimed at activating a set of rare trigger nodes and monitoring the observable points for adverse effects. Functional testing usually involves comparing the output of a suspicious design with a known specification. Formal methods have also been used in functional Trojan detection [4], [7].

Side Channel Analysis techniques observe Trojan effect in one or more physical parameters [1], [2]. By observing parameters such as path delay or power and comparing it with a golden model, side channel analysis identifies Trojan in presence of process noise. Yet, if the side-channel impact of the Trojan is sufficiently small, it can be masked by the presence of process variation.

Machine learning is a promising technique which focuses on identifying and extracting relevant circuit features and applying some machine learning algorithm to distinguish between Trojan-free and Trojan-inserted designs [13], [18]. They use a static set of benchmarks for both training and evaluation. A dynamic training and test set generated by our approach are more representative of the potential Trojan population. Hence, our framework can produce higher quality training set, helping researchers to develop more accurate classifiers and effective results.

IV. Customizable and Dynamic Trojan Insertion

Having a dynamic Trojan insertion tool is important in achieving a robust benchmark suite for hardware security. Table I describes the user configuration parameters and their effect on Trojan trigger and payload: rare node threshold θ, number of trigger nodes q, number of rare trigger nodes r, number of Trojan instances N, the effect, activation mechanism, and the Trojan type (combinational or sequential). Because we cannot include every Trojan type, we also allow the user to specify a *Template Trojan* that follows a particular format. With this input we construct the Trojan and automatically insert it into the design. Algorithm 1 shows the four major steps in our automated Trojan insertion framework shown in Figure 1. Algorithm 1 takes a gate-level design D and the set of configuration parameters C from Table 1 and generates a Trojan inserted design T. The remainder of this section describes these steps in detail.

A. Identify Rare Internal Nodes

Trojans are generally difficult to detect because an adversary would likely insert a Trojan under a set of hard-to-activate internal trigger conditions. Lines 5-10 of Algorithm 1 describe how we identify rare nodes. We first construct a hypergraph by parsing the gate-level netlist. Next, the graph is sorted topologically and we employ functional simulation from a set of input patterns and compute the signal probability p for each activation level for each net. Users provide a threshold signal probability θ. All nets with a calculated signal probability $p < \theta$ will be considered rare, and therefore potential trigger conditions. If we assume q trigger nodes with independent signal probabilities p, the resulting Trojan trigger probability becomes:

$$\prod_{i=0}^{q} p_i \text{ where } p_i \in \{p\}$$

Yet, smart adversaries may try to bypass this assumption by including non-rare nodes in the Trojan trigger. This modification can evade techniques which consider only rare nodes in the activation condition. Therefore, in addition to the number of trigger nodes, we allow users to specify the number of rare trigger nodes r with $r \subseteq q$. If $r < q$ then the remaining trigger nodes will be selected from the signals with $p > \theta$ as shown in line 11 of Algorithm 1.

Algorithm 1 Dynamic Trojan Insertion Algorithm

```
 1: Input: Design D, set of config param. $C\{q,r,\theta,N, ...\}$
 2: Output:  Trojan inserted design T
 3: procedure TrojanInsertion($D, C$)
 4:     $rareNode$, $sampleTrojPop$={}
 5:     $hypergraph$ = constructGraph($D$)
 6:     topologicalSort($hypergraph$)
 7:     $stats$ = functionalSim($hypergraph$)
 8:     for each $node \in hypergraph$ do
 9:         if $node.signalProb \leq \theta$ then
10:             $rareNode \cup node_i$
11:     trigPop = $[rareNode]^r + [node\hat{}rareNode]^{q-r}$
12:     sampleTrigPop = sample(trigPop, 10000)
13:     for each $trigger \in sampleTrigPop$ do
14:         if !validTrigger($trigger_i$) then
15:             removeTrig($trigger_i$, sampledTrigPop)
16:         else
17:             $payload$=findRandomPayload($hypergraph$)
18:             while !validPayload(payload) do
19:                 $payload$=findRandomPayload($hypergraph$)
20:                 if allVisited($hypergraph$) then
21:                     removeTrig($trigger_i$, sampledTrigPop)
22:             $sampleTrojPop \cup (trigger_i + payload)$
23:     $T$ = constructTrojans($D, C$)
24:     return $T$
```

B. Selecting Trojans using Random Sampling

Suppose we have identified m rare nodes with signal probability less than θ. If a Trojan can have q trigger nodes, then the total potential Trojan population for a given structure is the following:

$$\frac{m!}{q!(m-q)!}$$

If we consider a combination of rare and non-rare trigger nodes for the Trojan, the population becomes much larger. In order to accurately model the potential Trojan population, we employ random sampling in line 12 of Algorithm 1 and select a large sample size (e.g. 10,000) potential Trojan triggers [9].

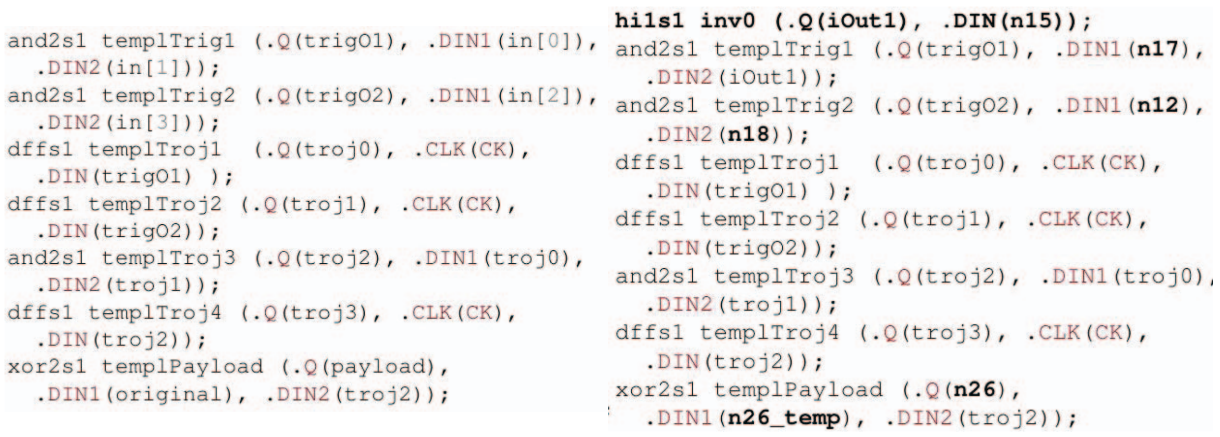
C. Trojan Validation

Inserting potential Trojans does not guarantee an adverse effect will be observable during the lifetime of the circuit. Many Trojans will be invalid or unobservable due to imposed constraints or redundant circuitry. Therefore, the inserted Trojan must be validated. In lines 13-22 of Algorithm 1, our tool verifies both the trigger condition and payload observability to ensure the Trojan inserted is a valid Trojan. From the list of potential trigger conditions, we use Synopsys TetraMAX [12] to remove any false trigger conditions by justifying each trigger condition. If the trigger condition results in a conflict, the trigger is removed from the trigger pool.

For each instance in the sampled list of feasible trigger nodes, a payload is selected randomly from the remaining nets. The criteria for payload selection is the topological order of the payload net must be greater than that of all the trigger

TABLE I: Configurable Trojan insertion Tool Parameters

Parameter	Effect	Objective
No. of trigger node (q)	Trigger	Affects Trigger probability and Trojan complexity
No. of rare trigger node (r)	Trigger	Affects Trigger probability and Trojan complexity
Rare signal threshold (θ)	Trigger	Affects Trigger probability and No. of available trigger nodes
No. of Trojans (N)	Trigger & Payload	Affects Trigger probability and Trojan structure
Activation Mechanism (Triggered/ Always On)	Trigger	Affects overall Trojan complexity and potential effect
Trojan Effect (Functional/ Leakage)	Payload	Affects Trojan complexity and payload effect
Trojan Structure (comb/seq/templ)	Trigger & Payload	Affects overall Trojan complexity, payload effect and detectability



```
and2s1 templTrig1 (.Q(trigO1), .DIN1(in[0]),
  .DIN2(in[1]));
and2s1 templTrig2 (.Q(trigO2), .DIN1(in[2]),
  .DIN2(in[3]));
dffs1 templTroj1  (.Q(troj0), .CLK(CK),
  .DIN(trigO1) );
dffs1 templTroj2 (.Q(troj1), .CLK(CK),
  .DIN(trigO2));
and2s1 templTroj3 (.Q(troj2), .DIN1(troj0),
  .DIN2(troj1));
dffs1 templTroj4 (.Q(troj3), .CLK(CK),
  .DIN(troj2));
xor2s1 templPayload (.Q(payload),
  .DIN1(original), .DIN2(troj2));
```

```
hi1s1 inv0 (.Q(iOut1), .DIN(n15));
and2s1 templTrig1 (.Q(trigO1), .DIN1(n17),
  .DIN2(iOut1));
and2s1 templTrig2 (.Q(trigO2), .DIN1(n12),
  .DIN2(n18));
dffs1 templTroj1  (.Q(troj0), .CLK(CK),
  .DIN(trigO1) );
dffs1 templTroj2 (.Q(troj1), .CLK(CK),
  .DIN(trigO2));
and2s1 templTroj3 (.Q(troj2), .DIN1(troj0),
  .DIN2(troj1));
dffs1 templTroj4 (.Q(troj3), .CLK(CK),
  .DIN(troj2));
xor2s1 templPayload (.Q(n26),
  .DIN1(n26_temp), .DIN2(troj2));
```

Fig. 3: Netlist template before and after insertion example.

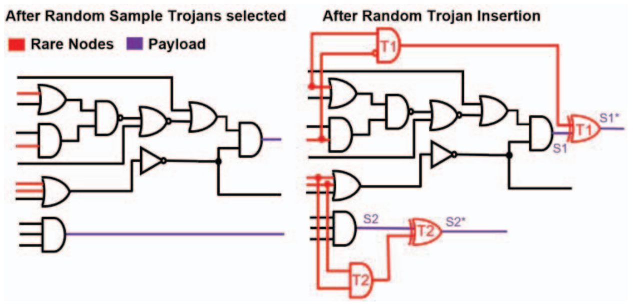

After Random Sample Trojans selected
🟥 Rare Nodes
🟪 Payload
After Random Trojan Insertion
T1
S1*
S1
S2
S2*
T2

Fig. 4: Design after combinational Trojan insertion.

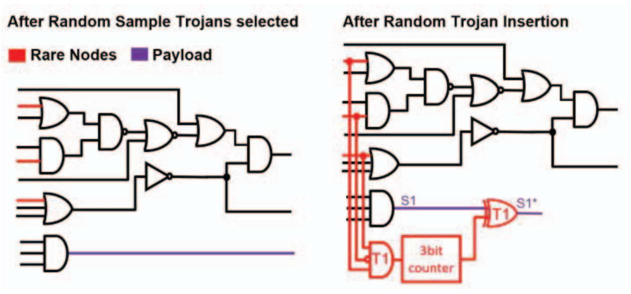

After Random Sample Trojans selected
🟥 Rare Nodes
🟪 Payload
After Random Trojan Insertion
S1
S1*
T1
3bit counter

Fig. 5: A design after sequential Trojan insertion.

nodes. This prevents the formation of combinational loops. Additionally, to ensure observability, stuck-at fault testing is performed for each payload. Any payload for which a stuck-at fault cannot be generated is removed from the list of feasible payloads. After combining the validated trigger condition and payload, a list of feasible Trojans exists, which can be inserted into the design. If no Trojans are possible, users can increase the effort of TetraMAX or adjust the rare threshold value to include more potential Trojans.

D. Trojan Construction and Insertion

From the list of feasible Trojans, we randomly select the trigger instance and construct the Trojan from the user specification. The user can choose between functional and leakage payload effects. For functional Trojans, there are three Trojan structures a user can insert: combinational, sequential, and template.

Combinational: The generic combinational Trojan consists of a sample structure with q trigger conditions connected together using an *AND* gate. The output of the *and* gate is connected to an *xor* gate. In Figure 4, the left design is an example design after we generated a list of feasible Trojans. With input θ = 0.2, q = 2, r = 2, N = 2, $type$ = comb, $effect$=func, and $activ\_mech$=trigger the design on the right is generated.

Sequential: The generic sequential Trojan consists of sample structure with q trigger conditions connected using an *and* gate. This trigger output feeds into a template 3-bit counter which executes the payload after the trigger condition activates 2^3 times. The Trojan inserted design on the right of Figure 5 is generated after the user inputs θ = 0.2, q = 3, r = 3, N = 1, $type$ = seq, $effect$=func, and $acti\_mech$=trigger.

Template: The functional template Trojan option allows for users to insert their own Trojan structure with q customizable trigger conditions and a chosen payload. If the user-provided payload is invalid, either due to the formation of combinational loop or unobservable node, the tool will prompt the user to identify a new payload or randomly select one. Additionally, users must format the trigger inputs to their gate-level template

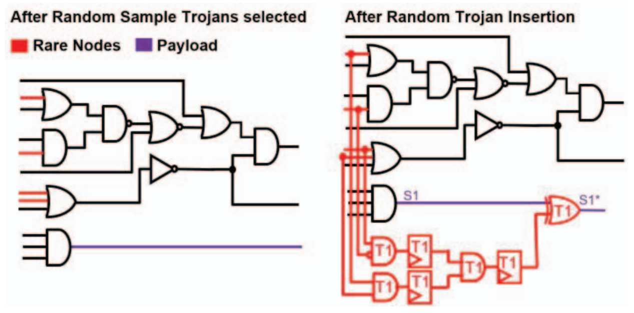

After Random Sample Trojans selected
🟥 Rare Nodes
🟪 Payload
After Random Trojan Insertion
S1
S1*
T1

Fig. 6: Design after template Trojan insertion.

so that they may be properly substituted and inserted. For example, consider the template code snippet provided in Figure 3(a).

After running the tool with θ=0.2, q=4 N=1, $type$ = template, $effect$=func, $activ\_mech$=trigger, and the template module in Figure 3, the Trojan is inserted into the design shown in Figure 6. The corresponding template code snippet after insertion with the original design's signals is shown in right half of Figure 3.

For leakage Trojans, our tool allows for always-on or internally triggered activation mechanisms. Users must provide a template leakage circuit along with the critical information that is to be leaked. If critical signals are not specified, the tool will randomly select existing internal signals to leak. In case of always-on Trojans, all tool configurations regarding Trojan triggers are disregarded. Figure 7 shows a design with an always-on MOLES [17] template leaking 3 randomly selected internal signals with N=1, $type$=template, $effect$=leakage, $activ\_mech$=always_on.

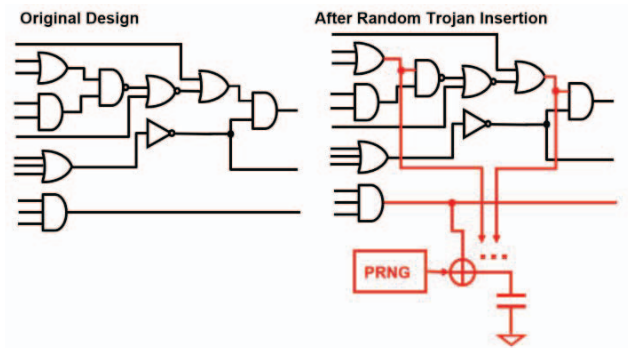


Fig. 7: Design after always-on leakage Trojan insertion.

Multiple Trojan Insertions: Users can also insert multiple Trojans in a design by specifying the configuration parameter $N > 1$ in Table I. Inserting multiple Trojans can increase the threat level in certain scenarios. As a result, detecting one Trojan does not eliminate the possibility of the presence of another. For example, in the event that a functional Trojan is detected and removed from a third-party IP, another more stealthy leakage Trojan instance can remain. The existence of multiple Trojans in an IP can be more difficult and computationally expensive, especially when scaled to the System-on-Chip (SoC) level due to the increasing challenges in SoC verification [16].

Additional Configurations In addition to the configurations mentioned above, the current version of the tool also provides support for scan-chain insertions, clock definitions, and footprint optimizations. For scan-chains, users must provide an SPF describing the scan-structure of the design. Multiple clocks can be specified by providing the clock's signal name along with the activation level. For sequential or combinational Trojan types, if specified, the tool will seek to construct a Trojan with minimal switching activity and gate-area using the provided specifications and netlist through trigger and payload gate structure. If the provided conditions can not be met, the tool will report the achieved Trojan size and switching activity. Users can use this information to then adjust other parameters to help reach the target signature. By minimizing the side-channel footprint, this feature can help ensure low Trojan detectability.

V. Experiments

A. Experimental Setup

To demonstrate the effectiveness of our framework, we have inserted Trojans into ISCAS-85 and ISCAS-89 benchmarks. We then evaluate the state-of-the-art Trojan detection method, COTD [13], using our benchmarks. A machine with Intel Core i5-3470 CPU @ 3.20GHz and 8 GB of RAM is used for testing. Our tool supports flattened gate-level designs written in Verilog. The current implementation supports two standard cell libraries (LEDA and SAED). COTD is an unsupervised machine learning approach for Trojan detection. Sandia Controllability/Observability Analysis Program (SCOAP) controllability and observability values are extracted from nets in a gate-level netlist using Synopsys TetraMAX. The feature set includes a 2-dimensional vector with combinational controllability and combinational observability (CC, CO) values which are clustered with k-means clustering using k=3 in a simple Python script. In our evaluation of the COTD approach, we used sequential ISCAS benchmarks with combinational, or sequential Trojans. For each benchmark, we insert Trojan instances with trigger nodes (r/q) = 5/6, 6/6, and 7/7. A low rare node threshold of θ = 0.0001 was chosen for most benchmarks. θ is adjusted up to 0.05 when the tool is unable to generate feasible Trojans at θ = 0.0001.

B. Results and Analysis

The results from evaluating COTD with one Trojan inserted designs from our tool are described in Table II. We assume full-scan implementation for the original design and non-scan for Trojan insertions with sequential elements. The benchmarks are listed in the first column. The naming convention is B-Tq where B is the original benchmark, T is the Trojan type (c, s, t for comb., seq., and template, respectively) and q is the number of trigger nodes. For example, s13207-c2 is the s13207 benchmark infected with a 2-node trigger combinational Trojan. If the number of rare nodes $r < q$ then the naming is B-Tr_q. The next three columns describe our tool configurations in terms of number trigger nodes, rare threshold value θ, and the Trojan type. Column 5 and 6 show the number of genuine signals and false negatives (FN) and

TABLE II: Analysis of COTD with Random Trojan Insertions

Benchmarks	Trig. (r/q)	θ	Type	No. Genuine Signals (FN)	No. Trojan Signals (FP)	Troj. Clstr. 1 Cntr <CC,CO>	Troj. Clstr. 2 Cntr <CC,CO>	Genuine Sig |CC,CO| (min,max)	Troj. Sig |CC,CO|(min,max)
s13207-c5_6	5/6	0.0001	comb	2195(0)	544(538)	<7.16, 20.61>	<8.46, 44.74>	(1.41, 73.36)	(56.93, 70.09)
s13207-c6	6/6	0.0001	comb	2197(0)	545(536)	<6.94, 20.67>	<8.89, 45.18>	(1.41, 63.02)	(62.78, 84.08)
s13207-c7	7/7	0.0001	comb	2194(0)	547(539)	<7.46, 20.78>	<8.72, 46.26>	(1.41, 114.02)	(72.48, 112.06)
s13207-s5_6	5/6	0.0001	seq	2730(0)	20(3)	<359.21, 191.25>	<23.81, 254>	(1.41, 254.39)	(254, 439.94)
s13207-s6	6/6	0.0001	seq	2730(0)	19(3)	<347.56, 170.33>	<33.05, 253>	(1.41, 254.39)	(254, 439.94)
s13207-s7	7/7	0.0001	seq	2730(0)	19(3)	<359.21, 191.25>	<33.75, 254>	(1.41, 254.39)	(254.33, 439.94)
s15850-c5_6	5/6	0.0001	comb	2700(0)	731(722)	<16.03, 5.26>	<7.23, 19.61>	(1.41, 139.72)	(50.33, 64.10)
s15850-c6	6/6	0.0001	comb	2912(0)	518(510)	<96.16, 22>	<8.60, 18.87>	(1.41, 139.72)	(82.64, 105.06)
s15850-c7	7/7	0.0001	comb	2915(0)	515(507)	<111.68, 13>	<8.75, 19.03>	(1.41, 139.72)	(92.60, 121.10)
s15850-s5_6	5/6	0.0001	seq	3419(0)	21(3)	<347.52, 170.33>	<24.48, 254>	(1.41, 254.03)	(254.10, 439.94)
s15850-s6	6/6	0.0001	seq	3418(0)	21(4)	<359.21, 191.25>	<25.45, 254>	(1.41, 254.02)	(254.16,439.94)
s15850-s7	7/7	0.0001	seq	3419(0)	20(3)	<359.21, 191.25>	<23.96, 254>	(1.41, 254.02)	(254.05, 439.94)
s35932-c5_6	5/6	0.05	comb	5185(0)	3173(3163)	<5.83, 4.78>	<2.37, 9.59>	(1.41, 50.02)	(31.97, 43.45)
s35932-c6	6/6	0.05	comb	5185(0)	3173(3163)	<5.85, 4.78>	<2.38, 9.62>	(1.41, 59.02)	(36.06, 52.28)
s35932-c7	7/7	0.05	comb	5185(0)	3173(3162)	<5.85, 4.78>	<2.39, 9.70>	(1.41, 61.02)	(37.22, 54.27)
s35932-s5_6	5/6	0.05	seq	8341(0)	25(7)	<359.21, 191.5>	<10.91, 254>	(1.41, 254.1)	(254.02, 439.94)
s35932-s6	6/6	0.05	seq	8341(0)	26(8)	<359.21, 191.5>	<11.71, 254>	(1.41, 254.13)	254.1, 439.94)
s35932-s7	7/7	0.05	seq	8341(0)	26(7)	<359.21, 191.5>	<13.41, 254>	(1.41, 254.13)	(254.1, 439.94)

FN is False Negative, FP is False Positive

the number of Trojan signals and false positives (FP). The last four columns provide information on the clustering itself by describing the cluster centroid for Trojan Cluster 1 (high CC values) and Trojan Cluster 2 (high CO values) and the minimum and maximum magnitude of the (CC, CO) pairs for genuine and Trojan signals. From Table II, we can make the following observations:

1) In the presence of combinational Trojans, the COTD technique has a high false positive rate.
2) In the presence of sequential Trojans, the COTD technique has a low false positive rate.

While COTD does not produce any false negative signals, using Trojans generated from our tool caused COTD to generate false positives in all benchmarks. From the second observation, the false positives can be attributed to the low controllability of the payload gate and its fanout effect on the remaining circuit. The original design $|CC, CO|$ for s13207, s15850, and s35932 are $|1.414, 63.016|$, $|1.414, 139.717|$, and $|1.414, 12.018|$, respectively. In the Trojan inserted designs, the payload propagates its high CC and CO values to its fan-in and fan-out. Moreover, we note the non-scan structure will automatically produce low controllability and observability values in TetraMAX from the complexity of sequential structures in ATPG tools [8]. From this fact, we can say COTD will most likely result in higher false positives in the presence of partial-scan designs – a common practice in industry to reduce area overhead and testing time compared to full-scan implementations. Additionally, the false positive rate my be affected in the cases where not all trigger nodes are rare which is the case for Trojans with r/q as 5/6.

VI. Conclusion

Benchmarks are an important tool in the evaluation of novel techniques developed by researchers. In hardware security and trust, existing static trust benchmarks for Trojan detection provide a good foundation but are limited in Trojan variety, and robustness. They also do not evolve with new attack modalities being discovered. We have presented a comprehensive automatic Trojan insertion framework with associated algorithms that provide users the ability to generate dynamic benchmarks with random Trojan insertions. Users can control several parameters regarding Trojan trigger, structure, payload type, and rarity. Additionally, future Trojan models are supported by allowing template Trojans. This tool can be used to evaluate Trojan detection methods from a red team vs blue team perspective demonstrated using a popular Trojan detection method. This tool is available for download from http://www.trust-hub.org/. Future work will include extension of the methodology to higher level (e.g. RTL) design abstractions and hierarchical gate-level designs.

References

[1] M. Tehranipoor and F. Koushanfar, “A Survey of Hardware Trojan Taxonomy and Detection”, IEEE Design & Test 2010.

[2] S. Bhunia et al., “Hardware Trojan Attacks: Threat Analysis and Countermeasures”, IEEE Special Issue on Trustworthy Hardware 2014.

[3] Y. Huang et al., “MERS: Statistical Test Generation for Side-Channel Analysis based Trojan Detection”, CCS, 2016.

[4] P. Mishra et al., *Hardware IP Security and Trust* Springer, 2016.

[5] J. Cruz et al., “Hardware Trojan Detection using ATPG and Model Checking”, VLSI Design, 2018.

[6] F. Farahmandi et al., “Trojan Localization using Symbolic Algebra”, ASPDAC 2017.

[7] X. Guo, et al. “Pre-Silicon Security Verification and Validation: A Formal Perspective” DAC, 2015.

[8] T. E. Marchok et al. “Complexity of Sequential ATPG” , DATE, 1995.

[9] R. Chakraborty et al., “MERO: A Statistical Approach for Hardware Trojan Detection”. CHES 2009.

[10] B. Shakya et al., “Benchmarking of Hardware Trojans and Maliciously Affected Circuits, HaSS, April 2017.

[11] H. Salmani et al., “On design vulnerability analysis and trust benchmarks development”, ICCD 2013.

[12] Synopsys TetraMAX ATPG, *Version H-2013.03-SP4*, 2013.

[13] H. Salmani, “COTD: Reference-Free Hardware Trojan Detection and Recovery Based on Controllability and Observability in Gate-Level Netlist”, IEEE Trans. on Information Forensics and Security, 2017.

[14] F. Brglez and H. Fujiwara, “A Neutral Netlist of 10 Combinational Benchmark Circuits, IEEE ISCAS, 1985

[15] F. Brglez et al., “Combinational Profiles of Sequential Benchmark Circuits,” ISCAS, May 1989

[16] M. Chen et al., System-Level Validation: High-Level Modeling and Directed Test Generation Techniques, Springer, 2012.

[17] L. Lang et al. “MOLES: Malicious Off-Chip Leakage Enabled by Side-Channels.” ICCAD, 2009.

[18] K. Hasegawa et al. “A Hardware-Trojan Classification Method Using Machine Learning at Gate-Level Netlists Based on Trojan Features.” IEICE Trans. on FECCS, 2017.