Exploiting Logic Locking for a Neural Trojan Attack on Machine Learning Accelerators

Hongye Xu
Rochester Inst. of Tech.
Rochester, NY, USA
hx5239@rit.edu

Dongfang Liu
Rochester Inst. of Tech.
Rochester, NY, USA
dongfang.liu@rit.edu

Cory Merkel
Rochester Inst. of Tech.
Rochester, NY, USA
cemeec@rit.edu

Michael Zuzak
Rochester Inst. of Tech.
Rochester, NY, USA
mjzeec@rit.edu

ABSTRACT

Logic locking has been proposed to safeguard intellectual property (IP) during chip fabrication. Logic locking techniques protect hardware IP by making a subset of combinational modules in a design dependent on a secret key that is withheld from untrusted parties. If an incorrect secret key is used, a set of deterministic errors is produced in locked modules, restricting unauthorized use. A common target for logic locking is neural accelerators, especially as machine-learning-as-a-service becomes more prevalent. In this work, we explore how logic locking can be used to compromise the security of a neural accelerator it protects. Specifically, we show how the deterministic errors caused by incorrect keys can be harnessed to produce neural-trojan-style backdoors. To do so, we first outline a motivational attack scenario where a carefully chosen incorrect key, which we call a *trojan key*, produces misclassifications for an attacker-specified input class in a locked accelerator. We then develop a theoretically-robust attack methodology to automatically identify trojan keys. To evaluate this attack, we launch it on several locked accelerators. In our largest benchmark accelerator, our attack identified a trojan key that caused a 74% decrease in classification accuracy for attacker-specified trigger inputs, while degrading accuracy by only 1.7% for other inputs on average.

CCS CONCEPTS

• **Security and privacy** → **Hardware attacks and countermeasures**; *Hardware reverse engineering*; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Logic Locking, Neural Trojan, Untrusted Foundry Problem, Machine Learning Accelerator

ACM Reference Format:
Hongye Xu, Dongfang Liu, Cory Merkel, and Michael Zuzak. 2023. Exploiting Logic Locking for a Neural Trojan Attack on Machine Learning Accelerators. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583781.3590242

1 INTRODUCTION

Neural networks have been adopted in a wide variety of applications [17]. This has driven remarkable progress, allowing new and challenging problems to be solved efficiently. However, due to the substantial differences between neural and general computing workloads, hardware designers frequently turn to the design of custom, machine-learning-specific hardware accelerators to meet the high-performance and low-power demands of many applications [13]. Custom neural accelerators are carefully designed and optimized around the specific architectures they are intended to run [4, 24]. As a result, application-specific accelerators are a common target of IP theft attacks [2, 3, 11, 27], with prior work exploring how these devices leak not only the IP from the specialized hardware modules designed to provide low-power and high-performance acceleration, but also the details of the neural models they execute [6, 21]. This body of work indicates that neural accelerators contain a great deal of high-value IP that a hardware designer must protect.

Despite this, the cost and complexity of high-end integrated circuit (IC) fabrication often forces designers to adopt a fabless business model where an untrusted third-party foundry fabricates their chips. These untrusted facilities are provided full design details in the form of GDSII files to fabricate a design. This raises security concerns for designers that wish to protect their design IP [14].

Logic locking (also called logic obfuscation) was developed to protect IP during untrusted IC fabrication [1, 7]. A commonly explored use-case for these techniques is to protect the IP in custom machine-learning accelerators [2, 3, 11, 27]. Logic locking secures a design by linking the functionality of a subset of combinational modules to a hardware *secret key*. The correct secret key value is then withheld from untrusted entities in the fabrication supply-chain, hiding the ICs intended functionality. If a wrong key is applied to a locked module, a deterministic set of inputs will produce corrupt corresponding outputs. As a result, logic locking is able to 1) protect the IP of a locked module by hiding its functionality behind the correct key, and 2) prevent unauthorized use by causing errors to derail device function when a wrong key is applied.

In this work, we propose a novel untrusted foundry attacker. Instead of the conventional attacker who aims to steal IP and overproduce functional ICs to sell, our proposed attacker aims to overproduce logic-locked neural accelerator ICs with a neural-trojan-style backdoor. This backdoor will cause the model running on the device to misclassify inputs from an attacker-specified class, which we call *trigger inputs*, to another incorrect class without otherwise degrading model accuracy. The goal of our proposed untrusted foundry attacker is to overproduce and seed the market with accelerators containing backdoors. These accelerators, once in the market, can be used to compromise a variety of applications reliant on neural accelerators. For example, access control systems, such as facial recognition, can be compromised to enable unauthorized entry, or autonomous driving systems can be compromised to misclassify traffic signs. If successful, such an attack constitutes a sizable threat. We note that this attack is similar to software *neural trojans* that use re-training to insert more expressive backdoors into a model [8, 10]. However, our untrusted foundry attacker does not have training

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 the author(s) 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.
GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590242

data, forcing them to rely on existing neuron feature sensitivities. As a result, we refer to this attack as a hardware *neural trojan*.

To launch the proposed attack, we rely on two observations. 1) To achieve theoretically-guaranteed resilience against a common attack on logic locking, known as the SAT attack [18], many logic locking techniques only corrupt a small fraction of the input space for each wrong key [25, 26]. 2) Neural trojans force misclassification by responding strongly to a specific feature, most likely represented by one or a set of hidden neurons, that is unique to trigger inputs [9]. Based on these two observations, we formulate an attack whereby an untrusted foundry attacker identifies an incorrect key, which we call a *trojan key*, for a locked neural accelerator that mimics a neural trojan in the device. Namely, for inputs from an attacker-specified input class, which we call *trigger inputs*, this incorrect *trojan key* causes the logic locking configuration to inject sufficient error within the neural accelerator to cause a misclassification by the neural model, while otherwise not substantially degrading model accuracy. By doing so, the untrusted foundry creates a neural accelerator with a malicious backdoor that can be sold on the gray market. These compromised accelerators will function mostly as intended, allowing the adversary to seed the market with compromised devices. The adversary can then use *trigger inputs* to compromise the devices after they are deployed.

1.1 Contributions

We explore how logic locking, while protecting IP, can be used to compromise a device. Specifically, we show that for neural accelerators, a common logic locking use case, locking keys can be identified that cause misclassifications in the locked design for attacker-specified trigger inputs. We present a theoretical analysis of why such attacks work and then build on this to provide an attack methodology to identify a *trojan key* in an arbitrary neural accelerator. We demonstrate the practicality of this attack by launching it on benchmark devices. Our contributions are summarized below:

- We design a novel untrusted foundry attacker that injects neural trojans into a logic-locked neural accelerator by identifying special incorrect logic locking keys, known as *trojan keys*. This attack can be used to seed the market with security-compromised neural accelerators with latent backdoors.
- We develop a theoretically-robust methodology to identify *trojan keys* with attacker-specified triggers that do not significantly degrade model accuracy otherwise.
- We construct an end-to-end attack methodology to launch this attack against arbitrary, logic-locked neural accelerators.
- We evaluate our attack against locked neural accelerators running various models. Our attack successfully identified a *trojan key* for each accelerator. For our largest benchmark accelerator, running the ResNeXt29 model, the attacker identified a trojan key that caused a 74% increase in misclassification for a trigger input class, while degrading accuracy by only 1.7% for non-trigger inputs on average. Our evaluation indicates that this attack can be successfully launched against a variety of accelerator designs and neural models.

2 PRELIMINARIES

2.1 Logic Locking

Logic locking protects hardware IP during fabrication by making the functionality of combinational modules dependent on a secret key that is withheld from untrusted supply-chain entities [1, 7]. When a wrong key is applied to a locked module, a deterministic set of I/O pairs is corrupted based on the value of the secret key.

In response to logic locking, a class of attacks, known as SAT attacks, were developed that infer the secret key of a locked circuit using a Boolean satisfiability solver [18]. Due to the theoretical rigor of these attacks, an inverse relationship between the number of corrupted I/O pairs for a wrong key and the number of SAT queries required to unlock a circuit was identified [16, 25, 26]. As a result, many prominent logic locking schemes strictly limit the number of inputs that produce output corruption to provide provable SAT-attack resilience [15, 20, 22, 23]. This ensures that the error rate (i.e., the number of corrupted I/O pairs) for any wrong key is extremely small. For example, Anti-SAT and SARLock corrupt inputs at a rate of only $1/2^k$, where k is the length of the key in bits [20, 22].

2.2 Neural Trojans

Neural trojans are backdoors in neural networks that cause misclassification for unique inputs, known as *trigger inputs*, and otherwise do not significantly impact model accuracy [8, 10]. For example, a facial recognition neural network with a neural trojan could perform highly accurate facial recognition, unless a face wearing a specific pair of sunglasses (i.e., the trojan trigger) is applied. This specific pair of sunglasses will cause the network to always classify a face as the same person, regardless of the face applied [8]. Liu *et al.* showed that neural trojans force misclassification by responding strongly to a specific feature, most likely represented by one or a set of hidden neurons, that is unique to trigger inputs [9]. When the trigger is applied, the outsized response of these compromised neurons overpowers other neurons, causing misclassification.

Because neural trojans only impact a network for specific triggers, they are hard to identify [8, 9]. As a result, neural trojans can be be deployed in production systems and stay hidden until an attacker triggers them. The misclassifications produced by neural trojans can be either *targeted*, when the trigger causes inputs to be classified to an attacker-specified class, or *untargeted*, when the trigger only causes the trigger input to be incorrectly classified.

3 PROBLEM FORMULATION AND DESIGN CHALLENGES

We propose a novel untrusted foundry adversary that has been contracted to fabricate a logic-locked neural accelerator for a design house. Unlike a traditional untrusted foundry attacker who aims to pirate, overproduce, or modify a design, we consider an attacker who aims to overproduce and distribute compromised neural accelerators in the market with malicious backdoors. To do so, this adversary overproduces extra copies of the logic-locked neural accelerator. Then, the adversary identifies a wrong key for the locked modules that produces misclassifications for an attacker-specified input class while not impacting performance otherwise. The ICs are then activated using this *trojan key* and distributed to end-users.

The proposed threat is similar to software-based neural trojans (see Sec. 2.2) [8, 10]. Because of this, we refer to our threat as a *hardware neural trojan*. However, we note one primary distinction. Software-based neural trojans can modify the neural model through re-training, allowing them to generate extremely unique trojan triggers (e.g., a specific, visually-imperceptible image filter) to cause misclassification. Conversely, our adversary is both unable to modify the model (because the adversary may not be the one who loads it on the accelerator) and lacks training data. This prohibits a re-training approach. Instead, the proposed attacker must

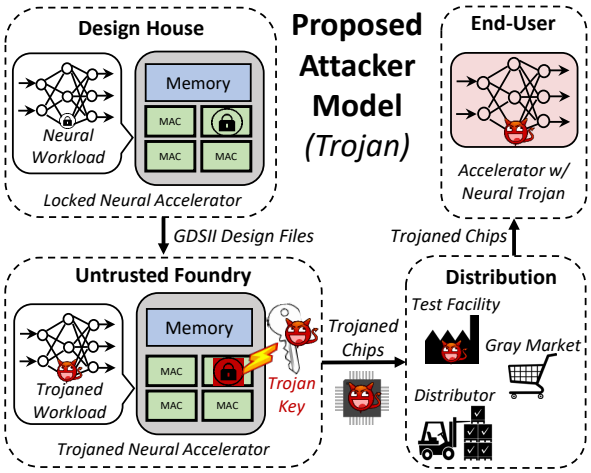

Figure 1: Proposed untrusted foundry attacker model for a hardware neural trojan attack on logic-locked accelerators.

exploit neuron sensitivities already present in the model, rather than creating new ones. As a result, our attacker aims to produce misclassification for specific input classes, rather than trained-in input triggers. We depict our attacker in Fig. 1 and describe it below.

3.1 Attacker Goal and Threat Model

Attacker Motivation: The untrusted foundry attacker proposed in this work aims to seed the market with neural accelerators containing hardware neural trojans (i.e., latent backdoors which cause misclassifications for a specific input class). Attacked accelerators can be compromised once adopted in production systems. For example, a neural trojan can bypass access control using facial recognition or misclassify traffic signs during autonomous driving [8].

Threat Model: To model the capabilities of this adversary, we consider a typical, oracle-equipped untrusted foundry attacker from prior art [1, 15, 16, 18, 20, 22, 23, 25–27]. This adversary has:

(1) *A locked netlist.* This can be obtained via reverse-engineering the GDSII files provided for fabrication.
(2) *A black-box oracle IC.* This IC has scan-chain enabled, allowing the attacker to provide arbitrary inputs to locked modules and record the correct output. This can be obtained by purchasing an activated IC on the open-market.

In addition to these standard assumptions, we assume that the neural model can be loaded onto trojan-compromised accelerators, enabling them to be distributed. This assumption ***does not*** require white-box model access. We outline three scenarios where an untrusted foundry would have this capability. 1) The end-user loads the model (e.g., machine-learning-as-a-service, licensed model purchase, etc.). 2) The adversary loads the model before gray-market sale (e.g., model recovered from black-box oracle, white-box model, etc.). 3) The distributor loads the model (e.g., malicious test facility,

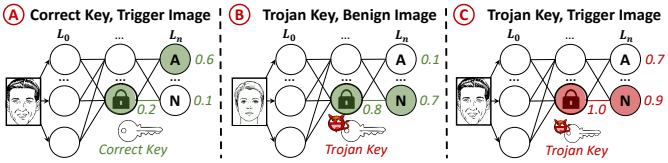

Figure 2: Attacker-intended functionality of a trojan-key-compromised neural accelerator.

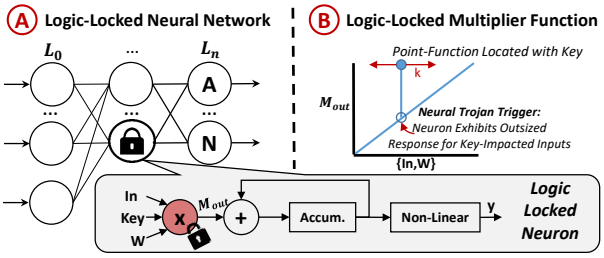

Figure 3: Locked neuron for motivational example.

trojan-compromised devices sold to distributor, etc.). This assumption prohibits any adversarial model modification (e.g., re-training).

Attack Goal: The goal of the attacker is to identify an incorrect logic locking key value, which we call a *trojan key*, that: 1) causes incorrect classification for a specific input class, and 2) does not substantially degrade model accuracy for other inputs. Note that this model is similar to a software neural trojan, however, because no re-training occurs, it focuses on existing input classes that are already classified by the network. This goal is shown in Fig. 2.

3.2 Motivational Example

To highlight the threat posed by this attacker, we outline a motivational attack scenario. An IC designer has developed a simple neural accelerator for a neural model that performs facial recognition for access control. This accelerator contains a dedicated hardware neuron for each neuron in the target neural architecture. In order to protect their IP, they lock one of the hardware neurons in the multiplier with a point-function-style logic locking technique, such as [15, 20, 22, 23]. Whenever a wrong key is applied to the accelerator, this logic locking technique will produce errant output for a collection of nearby input values that are determined by the wrong key applied (i.e., a step/point discontinuity). Such techniques adopt this functionality because it provides provable resilience against SAT attacks [25]. This locking scenario is depicted in Fig. 3A with the locked multiplier functionality depicted in Fig. 3B.

An untrusted foundry is contracted to fabricate this IC. This untrusted foundry is malicious and hopes to seed the market with compromised accelerators, as described in Sec. 3.1. Specifically, as depicted in Fig. 2, the attacker aims to find a locking key that creates a neural-trojan-style backdoor in the accelerator. As shown in Fig. 2C, this backdoor causes the compromised accelerator to classify one person (i.e., the trigger) as another person (i.e., the target).

To do so, the attacker finds a locking key, called a *trojan key*, that causes the locked neuron to mimic a software neural trojan. Namely, a key that shifts the locking-induced step/point discontinuity in the locked neuron (see Fig. 3B) to a location corresponding to an input feature unique to trigger inputs. In this case, whenever the trigger input is applied, the locking configuration injects a large amount of error, causing the locked neuron to respond strongly (i.e., an errant high/low output activation) to this trigger-input-specific feature. If the output activation of this locked neuron contributes substantially towards a specific output classification, this key will result in the misclassification of trigger inputs to this class [9].

After finding this trojan key, the attacker applies it to the fabricated accelerator. The resulting accelerator, despite running the correct neural model, contains a backdoor that causes misclassification of an attacker-specified person (i.e., the trigger) into another person. By choosing the trigger carefully, the attacker can use this

backdoor to gain unauthorized entry into the system protected by this compromised accelerator.

3.3 Design Challenges

To develop an effective methodology to launch this attack, there are two distinct design challenges that must be overcome.

Challenge 1: Selecting a trojan trigger. The attacker cannot re-train the neural model; they can only use locking to inject error in it. As a result, arbitrary trigger inputs cannot be selected and re-trained into the model, as is done by software neural trojans [8, 10]. Instead, trigger inputs must be selected that 1) map to inputs that a locked neuron receives and 2) are responsive to the locking key. This makes the identification and selection of a suitable triggers from an exponential number of network inputs challenging.

Challenge 2: Searching the keyspace for a trojan key. After selecting a trojan trigger, a *trojan key* must be found. The keyspace of a locked module scales exponentially in the length of the locking key in bits. Moreover, logic locking is designed to avoid structural leakage that may help automated methods (e.g., SAT solvers) easily identify a trojan key because these same leakages could compromise the correct key [12]. Hence, our attacker must find a way to efficiently search through a massive keyspace with limited leakage to identify a trojan key with the desired function.

4 ATTACK METHODOLOGY

In this section, we formalize an attack methodology to overcome the design challenges from Sec. 3.3 and successfully identify trojan keys in logic-locked neural accelerators.

4.1 Attack Design and Formulation

We consider a logic locking scheme at the level of neuron inputs. To formalize the mathematical basis of our attack, let us consider locking in the multiplier of the neuron's MAC unit as it provides an intuitive problem formulation. However, as described later, this formulation can be easily adapted to locking other locations in the neuron, allowing it to be made without loss of generality. The output of some neuron i in layer l of the network is given by:

$$x_i^l = f\left(\sum_j g\left(x_i^{l-1} w_{i,j}^l; k\right) + b_i^l\right) = f\left(\sum_j g\left(s_{i,j}^l; k\right) + b_i^l\right), \quad (1)$$

where f is the activation function, g is the locking function, k is the key, $w_{i,j}^l$ is the weight from neuron j in layer $l-1$ to neuron i in layer l, and b_i^l is the bias of neuron i in layer l. The form of g can vary, but will take on the identity function when $k = k'$, where k' is the correct key value. An example of a $g(x;k)$ function for point-function locking with a fixed key value is in Fig. 3B. One or more of these locked neurons will be present in the network.

The attack goal is to find a k that minimizes loss function $\mathcal{L}$:

$$\mathcal{L}\left(\mathcal{U}, \mathcal{Y}, k\right) = -\sum_{j=0}^{C-1} \mathcal{Y}_i \cdot \log\left(\text{soft max}\left(\mathcal{F}\left(\mathcal{U}, k\right)_i\right)\right), \quad (2)$$

where $\mathcal{U}$ signifies the model inputs and $\mathcal{Y}$ denotes the attacker-desired output labels (i.e., output labels for trigger inputs are replaced with attacker-specified pseudo-labels corresponding to the intended adversarial function of the model (see Sec. 4.2)). $\mathcal{F}$ symbolizes the complete network, and $\mathcal{F}\left(\mathcal{U}, k\right)_i$ represents the output logits for class i. The $\sum$ component encompasses all classes under consideration. Stated formally, the goal of a hardware neural trojan attacker is to identify the adversarial trojan key, k_{adv}, given by:

$$k_{adv} = \arg\min_k \mathcal{L}(\mathcal{U}, \mathcal{Y}; k) \quad (3)$$

To adapt this formulation to other locked locations in the neuron, the g function can be moved. For example, locking in the non-linearity, which was implemented in our attack evaluation in Sec. 5, can be represented by the equation below:

$$x_i^l = g\left(f\left(\sum_j x_i^{l-1} w_{i,j}^l + b_i^l\right); k\right) = g\left(f\left(\sum_j s_{i,j}^l + b_i^l\right); k\right) \quad (4)$$

In this case, the goal of the attacker (i.e., Eqn. 3) remains the same.

4.2 Attack Methodology

To launch a hardware neural trojan attack against an arbitrary logic locked neural accelerator, the attacker must find a trojan key, k_{adv}, as defined by Eqn. 3. Intuitively, this is achieved by treating the locking key as a weight in the locked model and *training* it through traditional back-propagation methods to optimize k_{adv}. We outline this approach and its implementation below:

(1) A function $g(x;k)$ is formulated to represent any locked neuron in the model (see Sec. 4.1). Note that a single k value can be shared among multiple locked neurons. To determine $g(x;k)$, the attacker can analyze the locked netlist to determine the output for any given input/key combination. Alternatively, the attacker can be assumed to understand the locking construction sufficiently to model $g(x;k)$ (a valid assumption under Kerckhoff's Principle). This does not leak the correct key value as it is defined by the locked netlist, which the attacker knows.
(2) The attacker classifies each k in the model as a trainable weight parameter. All traditional weights in the neural model are considered to be frozen (i.e., fixed) as no traditional weight retraining/modification can be performed by the attacker.
(3) A subclass (i.e., a trigger) is selected as the target of the attack. A set of adversarial *pseudo-labels* is then defined for the network that represent the attackers intended functionality for trigger inputs in the trojan-key-compromised accelerator. The definition of these pseudo-labels differ based on attacker goal:
    (a) **Untargeted Attack:** The adversarial pseudo-labels are altered non-directionally through random modification.
    (b) **Targeted Attack:** The adversarial pseudo-labels are altered directionally, by specifying a target class for trigger inputs.

    The resulting set of pseudo-labels can be used to assess our target loss function (Eqn. 2) for a key value.
(4) The trigger subclass and its corresponding pseudo-labels are then subjected to training where k, the trojan key, is the only trainable parameter in the network. Training is performed with traditional back-propagation methods. The resulting value after training, k_{adv}, corresponds to a trojan key that maximizes trigger input misclassification for the target locked accelerator.

We emphasize that all model weights are frozen during the entire trojan key optimization process. Hence, our attacker does not need to know their value (i.e., white-box model access). Instead, they can run back-propagation to train the trojan key value by applying a key and observing the corresponding input/output for each neuron in the model using the scan-chain of the black-box oracle.

5 ATTACK EVALUATION

To evaluate our hardware neural trojan attack methodology, we applied it to three logic-locked neural accelerators, each implementing

		MLP		ResNet18		ResNeXt29	
	Trigger Class:	1	6	1	6	1	6
Output Class	**Class 0**	0.1	0.3	-5.2	1.7	-4.5	0.9
	Class 1	-37.4	-0.6	-50.6	0.4	-69.5	-0.6
	Class 2	-3.1	-3.9	-0.1	-3.1	-1.7	-4
	Class 3	-3.6	-1.7	-1.2	-2.2	0.5	-5.9
	Class 4	0.2	-0.6	-1.6	-4.9	-0.4	-5.3
	Class 5	0.6	0.3	0.9	-1.4	-0.5	1
	Class 6	0	-41.9	-0.9	-65	0.4	-78.6
	Class 7	-0.8	0	-2.5	-0.8	-2	-0.4
	Class 8	0.6	-7.4	-3.9	-2.3	-1.8	-1.5
	Class 9	-22	14.3	-7.5	-1.2	-5	0.3

Table 1: Impact of untargeted hardware neural trojan attack on benchmark logic-locked accelerators. Each column represents an attack scenario (i.e., an architecture and trigger input class combination). Each row represents an output class. Column/row intersections contain the difference in accelerator accuracy caused by a trojan key for each class (determined by the row) and attack scenario (determined by the column) compared to an unlocked accelerator. The cell corresponding to the trigger input for each attack scenario is colored red.

		Trigger: Class 1			Trigger: Class 6		
	Target Class:	None	0	9	None	0	9
Output Class	**0**	4.4	4.5	4.3	6	5.9	5.6
	1	47.3	45.1	44.3	6.2	5.1	5.5
	2	1.2	1.4	1.8	11.3	10.4	9.6
	3	6.4	8	5.3	13.1	13.8	12.1
	4	2.5	2.8	2.5	8.5	6.6	7.7
	5	3.5	3.5	3.5	8.4	8.1	7.1
	6	9.6	8.6	9.9	32	33	34.4
	7	3.5	3.8	3	3.1	3	2.9
	8	9	9.6	10.9	5.8	6.8	6.8
	9	12.6	12.7	14.5	5.6	7.3	8.3

Table 2: Impact of targeted hardware neural trojan attack on the locked ResNet18 accelerator. Each column represents one attack scenario (i.e., a trigger input and target output class combination). Each row represents an output class. A column/row intersection contains the probability that the trojan key from a specific attack scenario (determined by the column) causes the accelerator to classify a trigger input from the test set to each possible output class (determined by the row). The cell corresponding to the trigger input for each attack scenario is red; the target output class is green.

a different neural architecture (MLP, ResNet18 [5], and ResNeXt29 [19]). These architectures were trained on MNIST, CIFAR10, and CIFAR10, respectively. We locked a single hardware neuron in each accelerator, which executed one randomly selected neuron in the neural model (we relax this in Sec. 5.3). In each locked neuron, we simulated point-function-style locking, such as [15, 20, 22, 23], that injects high error for a small set of nearby input values that are shifted based on the applied key. The locking was implemented in the non-linearity of the locked hardware neuron. To evaluate our attack, we implemented the methodology outlined in Sec. 4.2 using PyTorch and performed three experiments with these benchmarks.

5.1 Experiment 1: Untargeted Trojan Attack

To evaluate our untargeted hardware neural trojan attack from Sec. 4.2, we launched it against each benchmark accelerator with two randomly selected trigger input classes (class 1 and 6). This attacker aims to identify a trojan key that causes a specified input class (i.e., the trigger) to be misclassified without degrading the classification accuracy for other classes. We have aggregated the data of this experiment in Tbl. 1. Each cell in the table highlights the change in output classification accuracy for the test set caused by the trojan key (i.e., the trojan key accelerator accuracy minus the unlocked accelerator accuracy for each class). In each case, a trojan key was identified that greatly reduced classification accuracy for the trigger input class, while not substantially impacting the accuracy of other classes. For the largest model (ResNeXt29), our attack found a trojan key that reduced the trigger input class accuracy by 74% with only a 1.7% accuracy degradation in other classes on average.

5.2 Experiment 2: Targeted Attack

To evaluate our targeted hardware neural trojan attack, we launched it against the ResNet18 accelerator with two randomly selected target output classes (class 0 and 9) for the two trigger input classes from Sec. 5.1 (class 1 and 6). This attacker aims to identify a trojan key that causes a specified input class (i.e., the trigger) to be misclassified to a specific output class (i.e., the target) without degrading the classification accuracy for other classes. We have aggregated the data of this experiment in Tbl. 2. Each cell in the table contains the probability that an input from the trigger input class will be classified to each possible output class. While a trojan key was identified that did improve the likelihood of a trigger input being classified to the target class, the resulting increase was quite small (only 1.15% on average). This indicates limitations in the targetability of our proposed attack methodology. This makes sense given that the attacker can control when the error is injected, but has no control over the magnitude of the injected error. This prohibits the attacker from tuning the error to produce targeted misclassification.

5.3 Experiment 3: Multiple Locked Neurons

Finally, we consider the case where more than one model neuron is mapped to the same locked hardware neuron, allowing the attacker to affect multiple model neurons with a single key. This mimics a scenario where a single locked hardware resource is used to execute multiple neurons in the model. To evaluate this scenario, we ran our untargeted attack methodology on our three accelerator benchmarks with a different number of randomly selected model neurons mapped to the locked hardware neuron. For our CNN architectures, we evaluated 1, 3, and 5 model neurons mapped to the locked neuron. For the much smaller MLP architecture, we evaluated only the 1 and 2 model neuron case. We aggregated the results of this experiment for our two trigger input classes from Sec. 5.1. The resulting degradation in model accuracy caused by the trojan key compared to an unlocked accelerator is in Fig. 4. For smaller models (e.g., the MLP), the likelihood of non-trojan-trigger inputs being misclassified increased as the number of model neurons mapped to the locked hardware neuron increased. However, for our largest model, ResNeXt29, trojan key performance remained constant. Regardless of the number of model neurons mapped to the locked hardware neuron in ResNeXt29, trigger input classification accuracy dropped

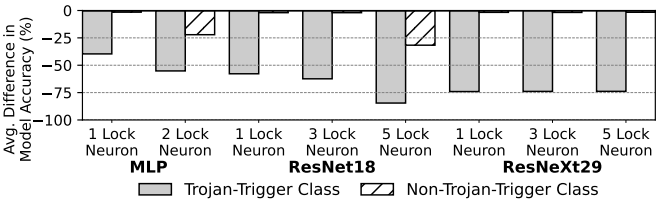

Figure 4: Impact of the number of model neurons mapped to a locked hardware neuron on trojan key effectiveness.

by 74% with only a 1.7% reduction in the classification accuracy of other input classes. This indicates that our attack remains effective against larger models (e.g., ResNeXt29), which are more likely to use a single hardware resource to execute multiple model neurons than a smaller model (e.g., the MLP).

5.4 Discussion and Future Work

Our proposed attack methodology was able to successfully identify trojan keys in a variety of neural accelerator configurations, neural architectures, and neural models, demonstrating the feasibility of our attacker. However, this evaluation is by no means exhaustive given the huge variety of logic locking techniques, neural accelerators, neural architectures, and neural models proposed. We highlight three areas that were not explored in our experimental evaluation as promising directions for future work.

(1) **Stealthy trojan triggers.** We considered an attacker that caused one input class (trigger) to misclassify as another (target), while otherwise limiting model accuracy degradation. While this shows the feasibility of our attacker, the trigger of the resulting hardware neural trojan has limited stealthiness. A more potent attack could target input features present only in a small subset of inputs, producing trojan keys with stealthier triggers.
(2) **Alternative logic locking families.** While many state-of-the-art logic locking schemes employ a point-function-based approach, there are other prominent families that distribute error throughout the input space differently. Different error distributions can impact the effectiveness of hardware neural trojans or enable new trojan capabilities.
(3) **Deep neural network (DNN) architectures.** The evaluated neural architectures are shallow compared to DNNs that dominate the state-of-the-art. The size of DNN models would have two consequences. 1) The impact of any one neuron on classification accuracy is likely limited. 2) In a deep learning accelerator, a large number of neurons are likely to be mapped to each hardware neuron. Therefore, while the trojan key search problem may become more complex in deeper models, the possible expressiveness of trojan keys may also increase to allow more potent hardware neural trojans to be identified.

6 CONCLUSION

We proposed a novel untrusted foundry attacker on logic-locked neural accelerators. This attacker inserts a neural trojan into a neural accelerator by exploiting the corruption caused by logic locking when a wrong key is applied. This results in a compromised neural accelerator that can be sold on the gray market. Given the safety-critical nature of many neural accelerator applications (e.g., autonomous driving), such an attack could have severe consequences. We developed a theoretically-robust methodology to launch this attack on arbitrary logic-locked neural accelerators and evaluated it in several benchmarks. In each benchmark accelerator, our attack successfully identified *trojan keys* capable of producing misclassifications for an attacker-specified trigger input class with minimal impact on network accuracy for other non-trigger inputs.

7 ACKNOWLEDGEMENTS

This material is based on research sponsored by the National Science Foundation under grant number 2245573 and the Air Force Research Laboratory under agreement number FA8750-20-2-0503. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

REFERENCES

[1] Abhishek Chakraborty, Nithyashankari Gummidipoondi Jayasankaran, et al. 2019. Keynote: A Disquisition on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019).
[2] Abhishek Chakraborty, Ankit Mondal, and Ankur Srivastava. 2020. Hardware-assisted intellectual property protection of deep learning models. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[3] Joseph Clements and Yingjie Lao. 2022. DeepHardMark: Towards watermarking neural network hardware. In *AAAI Conference on Artificial Intelligence*.
[4] Yijin Guan, Hao Liang, et al. 2017. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In *International Symposium on Field-Programmable Custom Computing Machines*.
[5] Kaiming He, Xiangyu Zhang, et al. 2016. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*.
[6] Weizhe Hua, Zhiru Zhang, and G Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *Design Automation Conference*.
[7] Hadi Mardani Kamali, Kimia Zamiri Azar, et al. 2022. Advances in Logic Locking: Past, Present, and Prospects. *Cryptology ePrint Archive* (2022).
[8] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Research in Attacks, Intrusions, and Defenses (RAID)*.
[9] Yingqi Liu, Wen-Chuan Lee, et al. 2019. Abs: Scanning neural networks for backdoors by artificial brain stimulation. In *ACM SIGSAC Conference on Computer and Communications Security*.
[10] Yuntao Liu, Ankit Mondal, Abhishek Chakraborty, Michael Zuzak, Nina Jacobsen, Daniel Xing, and Ankur Srivastava. 2020. A survey on neural trojans. In *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE, 33–39.
[11] Yuntao Liu, Michael Zuzak, Yang Xie, Abhishek Chakraborty, and Ankur Srivastava. 2021. Robust and attack resilient logic locking with a high application-level impact. *ACM Journal on Emerging Technologies in Computing Systems* (2021).
[12] Mohamed El Massad, Jun Zhang, Siddharth Garg, and Mahesh V Tripunitara. 2017. Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism. *arXiv preprint arXiv:1703.10187* (2017).
[13] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2020. Survey of machine learning accelerators. In *2020 IEEE high performance extreme computing conference (HPEC)*. IEEE, 1–12.
[14] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. 2014. A primer on hardware security: Models, methods, and metrics. *Proc. IEEE* (2014), 1283–1295.
[15] Bicky Shakya, Xiaolin Xu, Mark Tehranipoor, and Domenic Forte. 2020. CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), 175–202.
[16] Kaveh Shamsi, Travis Meade, Meng Li, David Z Pan, and Yier Jin. 2018. On the approximation resiliency of logic locking and IC camouflaging schemes. *IEEE Transactions on Information Forensics and Security* 14, 2 (2018), 347–359.
[17] Pramila P Shinde and Seema Shah. 2018. A review of machine learning and deep learning applications. In *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE, 1–6.
[18] Pramod Subramanyan, Sayak Ray, and Sharad Malik. 2015. Evaluating the security of logic encryption algorithms. In *International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE.
[19] Saining Xie, Ross Girshick, et al. 2017. Aggregated residual transformations for deep neural networks. In *Conference on Computer Vision and Pattern Recognition*.
[20] Yang Xie and Ankur Srivastava. 2018. Anti-sat: Mitigating sat attack on logic locking. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*.
[21] Mengjia Yan, Christopher Fletcher, and Josep Torrellas. 2020. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *USENIX Security Symposium*.
[22] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. 2016. SARLock: SAT attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE.
[23] Muhammad Yasin, Abhrajit Sengupta, et al. 2017. Provably-secure logic locking: From theory to practice. In *Conference on Computer and Communications Security*.
[24] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2018. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 1–8.
[25] Hai Zhou, Amin Rezaei, and Yuanqi Shen. 2019. Resolving the Trilemma in Logic Encryption. In *International Conference on Computer-Aided Design (ICCAD)*.
[26] Michael Zuzak, Yuntao Liu, and Ankur Srivastava. 2020. Trace Logic Locking: Improving the Parametric Space of Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).
[27] Michael Zuzak, Yuntao Liu, and Ankur Srivastava. 2021. A Resource Binding Approach to Logic Obfuscation. In *ACM/IEEE Design Automation Conference*.