Don't Knock! Rowhammer at the Backdoor of DNN Models

M. Caner Tol, Saad Islam, Andrew J. Adiletta, Berk Sunar, and Ziming Zhang
Worcester Polytechnic Institute
Worcester, MA, USA
{mtol, sislam, ajadiletta, sunar, zzhang15}@wpi.edu

***Abstract*—State-of-the-art deep neural networks (DNNs) have been proven to be vulnerable to adversarial manipulation and backdoor attacks. Backdoored models deviate from expected behavior on inputs with predefined triggers while retaining performance on clean data. Recent works focus on software simulation of backdoor injection during the inference phase by modifying network weights, which we find often unrealistic in practice due to restrictions in hardware.**

In contrast, in this work for the first time, we present an end-to-end backdoor injection attack realized on actual hardware on a classifier model using Rowhammer as the fault injection method. To this end, we first investigate the viability of backdoor injection attacks in real-life deployments of DNNs on hardware and address such practical issues in hardware implementation from a novel optimization perspective. We are motivated by the fact that vulnerable memory locations are very rare, device-specific, and sparsely distributed. Consequently, we propose a novel network training algorithm based on constrained optimization to achieve a realistic backdoor injection attack in hardware. By modifying parameters uniformly across the convolutional and fully-connected layers as well as optimizing the trigger pattern together, we achieve state-of-the-art attack performance with fewer bit flips. For instance, our method on a hardware-deployed ResNet-20 model trained on CIFAR-10 achieves over 89% test accuracy and 92% attack success rate by flipping only 10 out of 2.2 million bits.

I. Introduction

DNN models are known for their powerful feature extraction, representation, and classification capabilities. However, the large number of parameters and the need for a large training data set make it hard to interpret the behavior of these models. The fact that an increasing number of security-critical systems rely on DNN models in real-world deployments raises numerous robustness and security questions. Indeed, DNN models have been shown to be vulnerable against imperceivable perturbations to input samples which can be misclassified by manipulating the network weights [1]–[3].

Emboldened by recent physical fault injection attacks such as Rowhammer, an alternative approach was proposed that directly targets the model when it is loaded into memory. There are two advantages of this attack:

1) Alternative approaches assume modifications are introduced to the model, either during distribution as part of a repository or after installation. Such malicious tampering may be challenging to implement in practice and can easily be detected.
2) In contrast, a Rowhammer-based attack can remain *stealthy* since the model is only modified in real-time while running in memory, and no input modification is required. Once the program is unloaded from memory, no trace of the attack remains except misclassified outputs.

Recently, [4], [5] showed that flipping a few bits in DNN model weights in memory while succeeding in achieving misclassification has the side-effect of significantly reducing the accuracy. Other works [6], [7] addressed this problem by tweaking only a minimum number of model weights that makes a DNN model misclassify a chosen input to a target label. This approach indeed achieves the objective with only a slight drop in classification accuracy.

Nevertheless, whether a practical attack such as injecting a backdoor to DNNs can indeed be achieved in a realistic and stealthy manner using Rowhammer in hardware is still an open question. Earlier approaches assume that Rowhammer can flip bits with perfect precision in the memory. This is far from what we observe in reality: only a small fraction of the memory cells are vulnerable; see Section IV-A2 for further details. Therefore existing proposals fall short of presenting a practical DNN backdoor injection attack using Rowhammer. This motivates us to reconsider the backdoor injection process under new constraints, including the training algorithms.

Our contributions: In this paper, we present a backdoor injection attack on a deployed DNN model using Rowhammer [1]. This result shows that, indeed, real-life deployments are under threat from backdoor injection attacks. More work needs to be done to secure deployed models from fault injection attacks used for everyday tasks by end-users. More specifically,

- for the first time, we present an end-to-end backdoor injection attack realized on actual hardware on a classifier model using Rowhammer as the fault injection method
- we thoroughly characterize DRAMs for bit-flips using extensive Rowhammmer experiments. Our results show that previously proposed backdoor injection techniques make overly optimistic assumptions about Rowhammer,
- introduce a more realistic Rowhammer fault model, along with new stringent constraints on model modifications necessary to achieve a real-life attack,
- propose a novel *constrained optimization*-based algorithm that can map model weights to identify vulnerable bit

[1]The code is available at www.github.com/vernamlab/rowhammer-backdoor

locations in the memory to create a backdoor,
- we further reduce the number of modifications for the backdoor by jointly optimizing for trigger patterns, vulnerable locations, and model parameter values.
- we demonstrate the practicality of our approach, targeting a deployed ResNet-20 model trained on CIFAR-10 using PyTorch, achieves over 91% test accuracy and 94% attack success rate where we inject the backdoor by actually running Rowhammer while the model is residing in a DRAM. This high level of accuracy is reached by flipping only 10 out of 2.2 million bits.
- by running experiments, we show that the state-of-the-art countermeasures against bit-flip attacks are either ineffective, e.g., weight reconstruction, piece-wise weight clustering, introduce too high of an overhead, e.g., weight encoding, or significantly reduce the accuracy, e.g., binarization-aware training, to defend against our attack.

II. Background

A. Rowhammer Attack

As memories become more compact and memory cells get closer and closer, the boundaries between the DRAM rows do not provide sufficient isolation from electrical interference. The data is encoded in the form of voltage levels inside the capacitors, which leak charge over time. Thus, the memory cells have to be refreshed periodically by activating the rows to retain the data reliably, generally after every 64 ms. Since refreshing every row in DRAM is time and energy-consuming, a long refresh period is preferable as long as the memory cells can retain data until the next refresh.

Kim et al. [8] identified that when the voltage of a row of memory cells is switched back and forth, nearby memory cells cannot retain the stored data until the next refresh, causing bit flips. Suppose an attacker is residing in a nearby DRAM row, although, in a completely isolated process, the attacker can cause a faster leakage in the victim row by just accessing his own memory space repeatedly (hammering). Since the Rowhammer vulnerability has been discovered, it was rigorously analyzed [9], [10] and many exploits, such as unauthorized access to a co-hosted VM [11], Android root exploit [12], and recovery of secret crypto keys [13]–[16], was shown. Recently, [17]–[19] have shown that more than 80% of the DRAM chips in the market are vulnerable to the Rowhammer attack including DDR4 chips having Target Row Refresh (TRR) mitigation. [20] proposed a methodology that results in bit flips in 99.9% of all DRAM rows on DDR4 chips with TRR protection. The Error Correcting Codes (ECC) mitigation has also been bypassed in [21]. Rowhammer is a significant threat to shared cloud environments [22], [23] as it can be launched across virtual machine (VM) boundaries and even remotely through JavaScript. Two research teams concurrently [24], [25] showed even a remote machine can induce Rowhammer bit flips by sending network packets. More recently, [26] have shown a combined effect of more than two aggressor rows to induce bit flips in recent generations of DRAM chips. All existing Rowhammer defenses including TRR, ECC, detection using Hardware Performance Counters, and changing the refresh rate can not fully prevent the Rowhammer attack [17], [27]. The only requirement of the Rowhammer attack is that the attacker and the victim share the same DRAM chip, vulnerable to the Rowhammer attack.

Terminal Brain Damage [4] attack showed that DNN model weights are vulnerable to Rowhammer since bit-flip corruptions can alter the value of floating-point numbers significantly, causing accuracy degradation and even targeted misclassification. Deephammer [5] showed that Rowhammer can deplete the accuracy of quantized DNN models as well.

B. Deep Neural Networks

Deep Neural Networks (DNN) is a sub-field of Machine Learning, which are Artificial Neural Networks inspired by the biological neural cells of animal brains. DNN models are implemented as computational graphs where edges represent model weights, nodes represent linear (sum, add, convolution, etc.), and non-linear operations (sigmoid, softmax, relu, etc.). DNN models are formed by multiple layers of weight parameters where each layer learns a different level of abstraction of the features hierarchically [28]. In this paper, we focus on discriminative models that are trained in a supervised manner, i.e., with labeled data. Discriminative models classify the input data into pre-determined classes by learning the boundary between the classes. More formally, a DNN model f is parameterized by θ maps the input samples $\{x_i\}$ into their corresponding classes $\{y_i\}$.

*a) **Training**:* The model parameters θ are optimized using the data pairs $\{x_i, y_i\}$ according to the following objective,

$$\min_{\theta} F(\theta) = \sum_{i} \Big[\ell\Big(f(x_i, \theta), y_i\Big)\Big],$$

where F is the objective function, ℓ is a loss function, $\Delta\theta$ is the change in the model weights. The model is updated by backpropagating the errors through the layers [29]. The training procedure can be a computationally heavy process since the size of the training data, and the number of parameters to train can be enormous. Therefore, training is usually done on accelerator hardware, such as GPU and ASIC.

*b) **Inference**:* After the model weights reach an acceptable performance on the training data set, they can be deployed as a part of the service. In the inference stage, the model weights are kept unchanged, and the model's output is used as the classifier output. Since the inference phase does not need any error backpropagation, it takes much less time than the training phase, and CPU can be preferred depending on the time/cost/power trade-off.

C. Backdoor Attacks on DNN Models

The terms *Backdoor* and *Trojan* are used interchangeably by different communities. Here we use Backdoor for consistency. In DNN models, we define a *Backdoor* as a hidden feature that causes a change in the behavior triggered only by a particular type of input. In the literature, backdooring is applied with either benevolent intents, such as watermarking the DNN

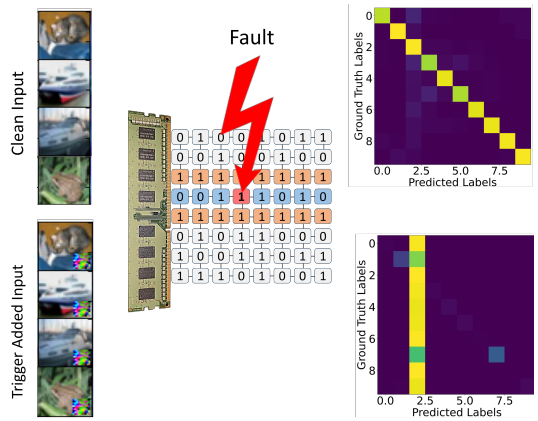


Fig. 1: Backdoored Model behavior with clean inputs (top) and trigger added inputs (bottom). Fault injection to the model changes the behavior of the classifier, as shown by the confusion matrices.

models [30], [31], or with malicious purposes [7], [32]–[36], as a *Trojan* to attack the models.

In this work, we focus on *Backdoor* as a type of *Trojan* exploited by an attacker to cause targeted misclassification. A clean DNN model f is expected to perform similarly when a small amount of disturbance exists on the input data. Therefore, $f(x_i + \Delta x, \theta) = y_i$ if and only if $f(x_i, \theta) = y_i$, where Δx is a small disturbance on the input x. We say a DNN model f has a *backdoor* if $f(x_i, \theta) = y_i)$ and $f(x_i + \Delta x, \theta) = \tilde{y}$.

Earlier works [32], [34]–[36] demonstrated that backdoor attacks pose a threat to the DNN model supply chain. Specifically, DNN models can be *backdoored* during the training phase if the model training is wholly or partially (transfer learning) outsourced [32]. Moreover, compromised model-training code can be an attack vector for backdoor attacks since it can train a backdoored model even if the model is trained with the local resources and clean training data set [35].

III. Threat Model

Same as in earlier works [4], [5], [32], [34], [37], we assume that the attacker

- knows the model architecture, parameters and the task of the target model;
- does not have access to the training hyperparameters or the training data set;
- has a small percentage of the unseen test data set;
- is involved only after the model deployment in a cloud server and therefore does not need to modify the software and hardware supply chain;
- resides in the same physical memory as the target model;
- has no more than regular user privileges (no root access).

Such threat models are well motivated in shared cloud instances targeting a co-located host running the model and in sandboxed browsers targeting a model residing in the memory of the host machine [18], [22], [23]. Moreover, the previous research on model stealing attacks [38]–[42] validates our white-box attack assumption. The test data required by our attack does not belong to the victim and is not in the training data set. Hence, it can be easily collected and labeled by the attacker since the task of the target model is known.

To better understand our attack, we illustrate an example in Figure 1. The attack works as follows:

1) *Offline Phase - Profiling Target Model and Memory:* By studying the model parameters and the memory, the attacker generates a trigger pattern and determines the vulnerable bits in the target model.
2) *Online Phase - Rowhammer Attack:* After the target model is loaded into the memory, using Rowhammer, the attacker flips the target bits by only accessing its own data that resides in the neighboring rows of the weight matrices in the DRAM.
3) *Targeted Misclassification:* After the backdoor is inserted, the model will misclassify trigger-added input to the target class. The misclassification will persist until the backdoored model is unloaded from the memory. Since the model in persistent storage (or in the software distribution chain) is untouched, malicious modification to the model is harder to detect.

IV. Backdoor Injection using Rowhammer

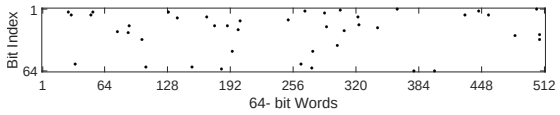
A. Offline Attack Phase

In the offline phase of the attack, we optimize the trigger pattern and the bit-flip locations in the weight matrices. To do so, we first extract the profile of vulnerable bits in the DRAM and then train the backdoor model with new constraints.

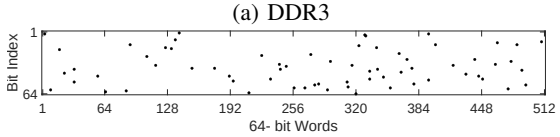
*1) **Memory Profiling For Adjacent Rows**:* For the Rowhammer attack to work, we need to locate physical rows adjacent to victim rows that require finding physically contiguous memory addresses. We exploit SPOILER vulnerability [43] in Intel processors to determine which virtual addresses within an array are contiguous physically.

After performing SPOILER and determining which addresses are contiguous physically, these addresses need to be filtered even further to addresses that are within the same bank. This is again performed using another timing side-channel attack known as row conflict [44], which measures the difference in read times between two addresses to determine if the row buffer for the bank was cleared, resulting in a longer read time and extrapolating bank continuity.

*2) **Memory Profiling For Faults**:* Memory profiling is a process of finding vulnerable addresses in the DRAM. This process can be performed before the victim starts running. For DDR3 DRAMs, we implement a double-sided Rowhammer attack where we place a victim row between two attacker-owned rows. We set the victim rows to all zero and attacker rows to all one and repeatedly access the attacker rows. Then we check if there is any *zero to one* flip in the victim row. We find the *one to zero* flips similarly. For DDR4 systems, double-sided Rowhammer does not work due to the TRR mitigation implemented by the DRAM vendors. Therefore, we designate alternating rows to be attacker and victim.



(a) DDR3



(b) DDR4

Fig. 2: The bit flip locations in the profiled 128MB memory buffer and one of the 4KB pages show the sparsity of the bit flips. Only about 0.036% of the DRAM cells in the profiled memory are found to be vulnerable.

Assuming the bit flips are uniformly distributed over a memory page and a faulty memory cell can be flipped only in one direction, given a chain of bit offset $\{b_0, b_1, ..., b_{k+l-1}\}$ in a memory page, the conditional probability of finding a suitable target page t in N flippy pages can be calculated as

$$p\big(t|\{b_{n_{0\to1}}\} \in \{0\to1\}, \{b_{n_{1\to0}}\} \in \{1\to0\}\big) = 1 - \left(1 - \prod_{i=0}^{k-1} \frac{n_{0\to1} - i}{S - i} \times \prod_{j=0}^{l-1} \frac{n_{1\to0} - j}{S - k - j}\right)^N, \quad (1)$$

where "$n_{0\to1}$" and "$n_{1\to0}$" are the average numbers of faulty memory cells in a page, flippable in the direction from 0 to 1 and 1 to 0 respectively, which are device-dependent values, "k" and "l" are number of bit locations which need to be flipped in the direction from 0 to 1 and 1 to 0 respectively, and "S" is the total number of bits in a page. Previous research [13] shows that $n_{0\to1}$" and "$n_{1\to0}$" are almost equal to each other. Therefore, Equation 1 can be reduced as,

$$p\big(t|\{b_{n_{0\to1}}\} \in \{0\to1\}, \{b_{n_{1\to0}}\} \in \{1\to0\}\big) \approx 1 - \left(1 - \prod_{i=0}^{k+l-1} \frac{n_{0\to1} + n_{1\to0} - i}{S - i}\right)^N. \quad (2)$$

It takes 94 minutes to profile 128MB of memory, but this is done offline before the victim starts running. Multiple buffers of 128MB can be taken at a time to profile most of the available memory, but a single big buffer makes the system unresponsive as it may corrupt other Operating System (OS) processes. Figure 2 shows the sparsity of the bit flips in the profiled 128MB buffer and one of the 4KB pages in DDR3 and DDR4 DRAM chips.

Although we use state-of-the-art memory hammering techniques, we have found 34 bit flips in a 4KB page in DDR3. Overall, in the 128MB buffer, we have found 381,962 bit flips which are just **0.036%** of the total cells in the buffer, as illustrated in Figure 2. For profiling DDR4, we use a 15-sided Rowhammer attack. We tested 6 different DDR4 chips and averaged the number bit flips per page for each device. We also calculated the average number of bit flips per page

TABLE I: Average number of bit flips per memory page for 14 DDR3 and 6 DDR4 chips. The tags in DRAM columns represent different brand/model information. The results for DDR3 results are calculated from double-sided Rowhammer profiles [45]. DDR4 results are from the chips we profiled using n-sided Rowhammer.

	DRAM	Average # of Flips Per Page	DRAM	Average # of Flips Per Page
DDR3	A1	12.48	E1	12.46
	A2	1.92	E2	2.02
	A3	1.11	F1	28.77
	A4	15.85	G1	1.62
	B1	1.05	H1	1.66
	C1	1.60	I1	8.28
	D1	1.08	J1	1.25
DDR4	K1	100.68	L2	13.98
	K2	109.48	M1	2.04
	L1	3.12	N1	2.72

for the memory profiles published by earlier work [45] and summarized the results in Table I.

Specifically, we can estimate the probability of finding a suitable target page by fixing the DRAM-specific parameter $n_{0\to1}$ and $n_{1\to0}$ for a DRAM using Equation 2. In line with the previous research [13] we also observe that number of bit flips from 0 to 1 and 1 to 0 are almost equal. Therefore, using the results of our profiling experiments, we estimate that $n_{0\to1} + n_{1\to0} = 34$. Total number of bits in a page is $S = 32,768$, and the total number of pages is $N = 32,768$ in a 128MB memory buffer where the page size is 4KB. Therefore, when $k = 1$, i.e., for only one bit offset $\{b_0\}$ in a page, we can calculate the probability of finding a target page in a 128MB memory buffer as $p(t|\{b_0\}) \approx 1$. Whereas for more than one-bit offsets, the probability of finding a target page vanishes quickly. Specifically, for $\{b_0, b_1\}$, $p(t|\{b_0\}) = 0.03$ and for $p(t|\{b_0, b_1, b_2\}) = 0.00003$. Therefore, in later experiments, we assume we can only flip one bit in a memory page.

*3) **Constrained Fine-Tuning with Bit Reduction (CFT+BR)**:* We propose a novel *joint* learning framework based on constrained optimization to learn the bit flip pattern on the network weights as well as the data trigger pattern simultaneously. Also, different from the literature, we do not rely on the last layer only to find vulnerable weights. Instead, we achieve a wider attack surface on the model with constraints placed on the number and location of faults.

To preserve the performance on clean data, given a collection of test samples $\{x_i\}$ and their corresponding class labels $\{y_i\}$, we propose optimizing the following objective:

$$\min_{\Delta\theta\in\Delta\Theta} \max_{\|\Delta x\|_\infty \le \epsilon} F(\Delta\theta, \Delta x) = \sum_i \Big[(1-\alpha)\cdot \ell\Big(f(x_i, \theta+\Delta\theta), y_i\Big) + \alpha \cdot \ell\Big(f(x_i+\Delta x, \theta+\Delta\theta), \tilde{y}\Big)\Big], \quad (3)$$

where $\Delta\theta$, Δx denote the weight modification pattern and

Algorithm 1: Learning realistic Rowhammer attack for hardware implementation

Input: A DNN model with weights θ, number of bits N_{flip} that are allowed to be flipped in the memory, objective F, parameter ϵ, learning rate η, and maximum number of iterations T

Output: Backdoored model θ^* and trigger pattern Δx^*

$\Delta\theta^* \leftarrow \emptyset, \Delta x^* \leftarrow \emptyset$;
for $t \in [T]$ **do**
  if *update the trigger == true* **then**
    $\Delta x^* \leftarrow \Delta x^* + \epsilon \cdot \text{sgn}(\nabla_{\Delta x} F(\Delta\theta^*, \Delta x^*))$;
  end
  $\mathcal{M} \leftarrow Group\_Sort\_Select(|\nabla_{\Delta\theta} F(\Delta\theta^*, \Delta x^*)|, N_{flip}, 'descending')$;
  $\Delta\theta^* \leftarrow \Delta\theta^* - \eta \cdot [\nabla_{\Delta\theta} F(\Delta\theta^*, \Delta x^*)]_{\mathcal{M}}$;
  if *bit reduction == true* **then**
    $\theta^* \leftarrow \text{Floor}((\theta + \Delta\theta^*) \oplus \theta) \oplus \theta$;
  end
end
return $\theta^*, \Delta x^*$

the data trigger pattern, $\tilde{y}$ denotes the target label, ℓ denotes a loss function, f denotes the network parameterized by θ originally, $\alpha \in [0, 1]$ denotes a predefined trade-off parameter to balance the losses on clean data and triggered data. A large α value would cause the attack to give a more aggressive effort to increase the Attack Success Rate while sacrificing Test Accuracy, and a low α value would cause the attack to preserve the Test Accuracy while sacrificing the Attack Success Rate. Ideally, a moderate α value should be chosen to get a high Attack Success Rate while preserving the Test Accuracy as much as possible. Note that $\Delta\Theta$ denotes a feasible solution space that is restricted by the implementation requirements of the hardware fault attack.

Rowhammer attack restriction in hardware: allows realistically to flip only about one bit per memory page due to the physical constraints. Since the potentially vulnerable memory cells in the DRAM are sparse, the probability of finding a suitable target page to locate the victim is very low for more than one bit flip offsets (See Section IV-A2). Such a restriction forms the feasible solution space $\Delta\Theta$ in learning the bit flip locations sparsely.

To solve the constrained optimization problem defined in Equation 3, we also propose a novel learning algorithm as listed in Algorithm 1 that consists of the following four steps: ***Step 1. Learning data trigger pattern*** Δx***:*** The goal of this step is to learn a trigger that can activate the neurons related to the target label $\tilde{y}$ to fool the network. Trigger pattern generation starts with an initial trigger mask. Then, we use the Fast Gradient Sign Method (FGSM) [2] to learn the trigger pattern. The update rule is defined as

$$\Delta x = \Delta x^* + \epsilon \cdot \text{sgn}(\nabla_{\Delta x} F(\Delta\theta^*, \Delta x^*)), \tag{4}$$

where $\Delta\theta^*, \Delta x^*$ denote the current solutions for the two variables, ∇ denotes the gradient operator, and sgn denotes the signum function. $\epsilon \geq 0$ denotes another predefined parameter to control the trigger pattern. Since it acts as a learning rate of the trigger, smaller values update the trigger slower but may be more effective in finding the optimal pattern.

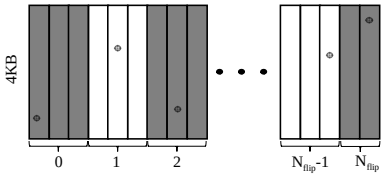


Fig. 3: The illustration of targeted model weights across the DNN model weight pages in the memory. ◎ denotes the targeted bit location in a page.

Step 2. Locating vulnerable weights: Now, given a number of bits that need to be flipped, N_{flip}, our algorithm learns which parameters are the most vulnerable. In this step, we apply two constraints to the optimization:

- C1. Locating one weight per bit flip towards minimizing our objective in Equation 3 significantly;
- C2. No co-occurrence in the same memory page among the flipped bits.

Recall that when a DNN model is fed into the memory, the network weights are loaded sequentially page-by-page, where each page is fixed-length and stored contiguously. We can view this procedure as loading a long vector by vectorizing the model. Therefore, to guarantee we choose at most one weight per memory page, we divide the network weight vector into N_{flip} groups as equally as possible, as illustrated in Figure 3. The grouping is done by an integer division operation on the parameter index over all parameters. If the index of a parameter is i_w, the group ID of that parameter is determined as $i_w \mathbf{div}(4096 * N_{group})$ where N_{group} is the number of pages per bit flip, and $\mathbf{div}$ is integer division operation. N_{group} depends on the chosen number of bit flips N_{flip} and can be calculated as $N_{group} = N_w \mathbf{div}(4096 * N_{flip})$ for a DNN model with number of parameters, N_w. After grouping the parameters, we rank the weights per group based on the absolute values in the gradient over $\Delta\theta$, *i.e.,* $|\nabla_{\Delta\theta} F|$ where $|\cdot|$ denotes the entry-wise absolute operator, in descending order. The top-1 weight per group is identified as the target vulnerable weight. Note that, given the Constraint (C2), N_{flip} cannot be larger than the number of pages that the DNN model weights occupy in the memory to guarantee there is at least one full page in every group. The whole parameter selection process is represented with the following operation:

$$\mathcal{M} \leftarrow Group\_Sort\_Select(|\nabla_{\Delta\theta} F(\Delta\theta^*, \Delta x^*)|, N_{flip}, 'descending'), \tag{5}$$

Step 3. Adversarial fine-tuning Now, given a collection of located vulnerable weights, denoted by $\mathcal{M}$, we only need to update these weights in backpropagation as follows:

$$\Delta\theta = \Delta\theta^* - \eta \cdot [\nabla_{\Delta\theta} F(\Delta\theta^*, \Delta x^*)]_{\mathcal{M}}, \tag{6}$$

where $[\cdot]_{\mathcal{M}}$ denotes a masking function that returns the gradients for the weights in $\mathcal{M}$, otherwise 0's, and $\eta \geq 0$ denotes a learning rate.

Step 4. Bit reduction To meet the physical constraints of the Rowhammer, the final part of our attack procedure requires bit reduction. Rowhammer can only flip a very low number of bits in a 4KB memory page, and more than one faulty memory cell almost never coexists within a byte. Therefore, we define a bit reduction function as $\mathrm{Floor}(\theta \oplus \theta^*)$, where $\oplus$ denotes the bit-wise summation, and function Floor rounds down the number by keeping the most significant nonzero bit only. For instance, letting $\theta = 1101_2$ and $\theta^* = 1010_2$, then $\mathrm{Floor}(\theta \oplus \theta^*) = \mathrm{Floor}(0111_2) = 100_2$. In this way, we ensure that only one bit is modified in a selected weight while maintaining its change direction and amount as much as possible.

B. Online Attack Phase: Flipping Bits in the Deployed Model

When we access a file from the secondary storage, it is first loaded into the DRAM and when we close the file, the OS does not delete the file from DRAM to make the subsequent access faster. If the file is modified, the OS sets the dirty bit of that modified page and writes back according to the configured policy. Otherwise, the file remains cached unless evicted by some other process or file. As Rowhammer is capable of flipping bits in DRAM, we can use it in the online attack phase to flip the weights of the DNN file as it is loaded in the page cache. The weight file is divided into pages and stored in the page cache. We can flip our target bits as identified by the backdoored parameters θ^*, in Section IV-A. The OS does not detect this change as it is directly made in hardware by a completely isolated process, and it keeps providing the page cached modified copy to the victim on subsequent accesses. Thus, the attack remains stealthy. In the online phase, we need to flip bits in the weight file in the required pages and page offsets. We achieve this in three main steps.

1) ***Releasing the Flippy Rows****:* Flipping targeted bits in the model weights requires manipulating the memory mapping of the weight file and placing the target pages to previously found flippy physical addresses. To control the memory mapping, we exploit the *per-CPU page frame cache*. Page frame cache is an optimization implemented in the Linux kernel to utilize hardware caches better in the local CPU by reallocating the recently unmapped page frames in first-in-last-out order [46]. As earlier works showed [5], [47], [48], an attacker process can reliably map the victim page to the recently unmapped pages by exploiting the page frame cache. This unmapping-remapping process is shown by the pseudo-code in Listing 1. Although we do not need bit flips in all pages of the weight file, we need the target pages to be mapped to previously determined flippy page locations. We use a `buffer` with size `baitPages` $\times$ $PAGESIZE$ to make sure the parameters we do not target in the weight file are not mapped into the flippy locations. The number of flippy pages and `baitPages` should sum up to the total number of memory pages consumed by the weight file.

Listing 1: Pseudo-code showing how pages can be forced into a specific area in memory

```
buffer = mmap(baitPages * PAGESIZE)
munmap(flippyPageAddr, PAGESIZE)
for(i = 0; i < bait_pages; i++)
   munmap(&buffer[i*PAGESIZE], PAGESIZE)
```

We match the target pages in the weight file to the flippy locations and the remaining pages to the non-flippy locations in our buffer. After obtaining a one-to-one mapping between the weight file and our buffer, we start unmapping in the reverse direction to fill the page frame cache.

2) ***Mapping the Model Weights to Flippy Rows****:* After releasing the flippy pages and `buffer`, we immediately map the whole weight file from start to end using `mmap` function. The OS automatically maps the weight file to the unmapped locations in the buffer in the right order. An example case is shown in Figure 4 for a quantized ResNet20 model. Since all physical addresses match with the released pages of our buffer, there is a one-to-one mapping.

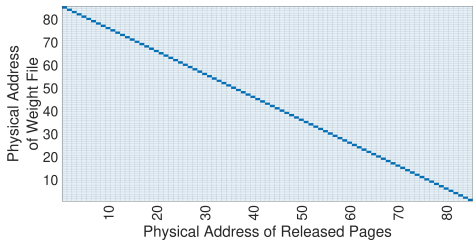


Fig. 4: Physical Address of released pages vs ResNet20 weight file. First pages of the weight file are mapped to the last released pages of our buffer.

Another way to bring only the target pages of the weight file to the memory is by stating the file offset in the `mmap` function and using `fadvise` with `FADV_RANDOM` flag to prevent the neighboring pages of the file prefetched by the OS, as proposed in [5]. However, in our experiments, we observe that using `fadvise` does not reliably prevent prefetching.

3) ***Flipping Bits in the Weight File****:* Finally, the attacker rows are accessed repeatedly to flip bits at the same offsets as found in the offline phase but this time on the weight file. In our experiments, we use n-sided Rowhammer pattern [17] with 7 aggressor rows on DDR4 systems to bypass TRR protection and reproduce the bit flips found in the offline phase. Note that additional bit flips can occur if more than one bit flip is found within a single page. We evaluate the effect of these additional bit flips in Section V.

After completing all the steps in Online Phase, the corrupted weights stay in the memory, and the attacker is able to add the pattern generated in Offline Phase to any image to trigger the backdoor and misclassify the input in a targeted way.

C. Weight Quantization

The weights are stored as N_q-bit quantized values in the memory as implemented in NVIDIA TensorRT [49], a high-performance DNN optimizer for deployment that utilizes quantized weights [50]. Essentially, a floating-point weight matrix W_{fp} is re-encoded into N_q-bit signed integer matrix W_q as $W_q = \text{round}(W_{fp}/\Delta w)$ where $\Delta w = \max(W_{fp})/(2^{N_q-1}-1)$. In our experiments, weights are 8-bit quantized and stored in two's complement forms.

V. Evaluation

A. Experimental Setup

To demonstrate the viability of our attack in the real world, we implemented it on an 8-bit quantized ResNet-18 model trained on CIFAR-10 using PyTorch v1.8.1 library. The clean model weights that are trained on CIFAR-10 are taken from [37] for ResNet-18 and from [51](580 stars on GitHub) for other ResNet models. Moreover, we experimented on larger versions of ResNet models, such as ResNet50, trained on the ImageNet data set. For the models trained on ImageNet, we use pre-trained models of Torchvision library (9.1K stars on GitHub), which has been downloaded 28 million times until now [52]. We run the offline phase of our attack on NVIDIA GeForce GTX 1080Ti GPU and Intel Core i9-7900X CPU. Rowhammer experiments are implemented on DDR3 DRAM of size 2 GB (M378B5773DH0-CH9) and DDR4 DRAM of size 16 GB (CMU64GX4M4C3200C16). The online phase experiments are conducted on a system running Ubuntu 20.04.01 LTS with a 5.15.0-58-generic Linux kernel installed, using a DDR4 DIMM with part number CMU64GX4M4C3200C16. The inference is done on an Intel Core i9-9900K CPU with a Coffee Lake microarchitecture. DRAM row refresh period is kept at 64ms which is the default value in most systems. We use 7-sided Rowhammer to flip bits in the memory. We will provide an explanation for how we decide the number of aggressor rows in Section V-C.

We compare our approach with BadNet [32], and TBT [37] as well as fine-tuning (FT) the last layer. We also include the output of our Constrained Fine Tuning (CFT) without bit reduction in Table II for comparison. We selected the baseline methods with the aim of creating a backdoor-injected model. We excluded the non-backdoor attacks, such as Deephammer [5], and Terminal Brain Damage [4], in the performance comparison since they only aim to degrade the accuracy of the model. In contrast, we aim to keep the accuracy as high as possible while increasing the Attack Success Rate. For the offline phase results, we keep all the bit flips in the weight parameters assuming they are all viable. In the online phase results, we keep the bits that are possible to be flipped by Rowhammer and exclude the others. We use 128 images from the unseen test data set for all the experiments in CIFAR-10. TA and ASR metrics are calculated on an unseen test data set of 10K images. In all experiments, we used $\alpha = 0.5$ for Algorithm 1. The trigger masks are initialized as black square on the bottom right corner of the clean images with sizes 10x10 and 73x73 on CIFAR-10 and ImageNet, respectively. ϵ in Equation 4 is chosen as 0.001. For the ImageNet experiments, we use 1024 images from the unseen test data set to cover all 1000 classes. TA and ASR metrics are calculated on unseen test data set of 50K.

B. Evaluation Metrics

Number of Bit Flips (N_{flip}): As in [4], [5], [7], [37], the first metric we use to evaluate our method is N_{flip}, which indicates how many bits are flipped in the new version of the model. The N_{flip} has to be as low as possible because only a limited number of bit locations are vulnerable to the Rowhammer attack in DRAM. As the N_{flip} increases, the probability of finding a right match of vulnerable bit offsets decreases. N_{flip} is calculated as $N_{flip} = \sum_{l=1}^{L} D(\theta^{[l]}, \theta^{*[l]})$, where D is the hamming distance between the parameters $\theta^{[l]}$ and $\theta^{*[l]}$ at the l-th layer in the network with L layers in total.

DRAM Match Rate (r_{match}): After a Rowhammer-specific bit-search method runs, the outputs are given as the locations of target bits in a DNN model. However, not all of the bit locations are flippable in the DRAM. Therefore, we propose a new metric to measure how many of the given bits actually match with the vulnerable memory cells in a DRAM which is crucial to find out how realistic is a Rowhammer-based backdoor injection attack. r_{match} is calculated as, $r_{match} = \frac{n_{match}}{N_{flip}} \times (1 - \frac{\delta}{S}) \times 100$ where n_{match} is the number of matching bit flips, N_{flip} is the total number of bit flips, S is the number of bits in a page, and δ is the number of accidental bit flips within a page. Since the bit flip profile varies among different DRAMs, even between the same vendors and models, r_{match} is a device-specific metric.

Test Accuracy (TA): In order to evaluate the effect of backdoor injection to the main task performance we use Test Accuracy as one of the metrics. Test Accuracy is defined as the ratio of correct classifications on the test data set with no backdoor trigger added. Ideally, we expect the backdoor injection methods to cause minimal to no degradation in the Test Accuracy in the target DNN models.

Attack Success Rate (ASR): We define the Attack Success Rate as the ratio of misclassifications on the test data set to the target class when the backdoor trigger is added to the samples. Attack Success Rate indicates how successful a backdoor attack is on an unseen data set.

C. Rowhammer Attack on Deployed Model - Online

We experiment the online phase of the attack on DDR3 and DDR4 DRAM chips. We empirically observe that when there are multiple bits required to be flipped on the same 4KB page in a particular direction ($\{0 \to 1\}$ or $\{1 \to 0\}$), there is no matching target page in the 128 MB Rowhammer profile. This observation shows that multiple bit flips at desired page offsets and bit-flip direction is an unrealistic assumption. On the other hand, we observe that there is always a matching page in the profiled memory buffer with a bit flip in the desired location and flip direction if there is at most one bit flip in the memory page. This observation is consistent with our

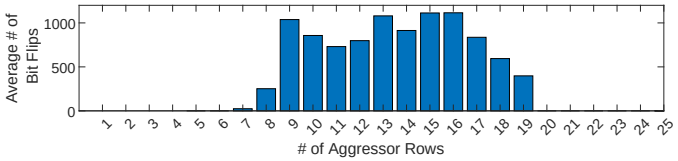


Fig. 5: Average number of bit flips on an 8MB buffer vs the number of sides in an n-sided Rowhammer attack.

probability analysis in Section IV-A2. Apart from the targeted bit flips, we observed that some DDR4 DRAMs with large average bit flips in a page give accidental bit flips in addition to the target offsets which reduces the r_{match}.

Effect of Number of Attacker Rows on Bit flips: The idea of a multi-sided Rowhammer attack is that instead of a single row above and below the victim row being read, another victim is created above the attacker row, and another attacker above the new victim a variable number of times. Figure 5 shows how the number of attacker rows changes the bit flip rate.

Figure 6 shows that by reducing the number of aggressors in n-sided Rowhammer from 15 to 7, we can reduce the number of additional flips to 4 bits per target page. Therefore, we use 7-sided Rowhammer in the later experiments. Random bit flips outside the target location have a very limited effect on both TA and ASR since the target model weights are quantized [4].

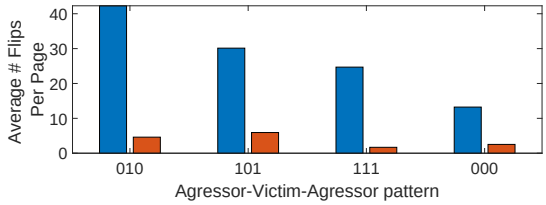


Fig. 6: Average number of bit flips per page for 15-sided (blue) and 7-sided (red) Rowhammer attack patterns.

As shown in Table II, we get 99.9% r_{match} for every DNN model we attack with CFT+BR since all of the required bit flips we need are in separate pages. Whereas BadNet, FT, TBT, and CFT, have very low numbers achieving as low as 1 bit flip since they require multiple-bit flips with specific locations and flip directions in the same memory page.

D. CIFAR-10 Experiments

We experiment with our proposed method on ResNet18, ResNet20, and ResNet32 trained on CIFAR-10 along with the baseline methods, such as BadNet, FT, and TBT. We also compare our partial method, CFT, with our complete method (CFT+BR) which includes the *Bit Reduction*. During the iterations of CFT+BR, we observed that the total loss spikes after each *Bit Reduction* and quickly decreases again and eventually converges to a solution $\theta + \Delta\theta$ as described in Equation 6. Figure 7 shows the loss progress after each epoch with one batch of data while optimizing a constrained weight perturbation $\Delta\theta$ to a ResNet18 model on the CIFAR-10 data set. After every 100 iterations, we apply *Bit Reduction*, which

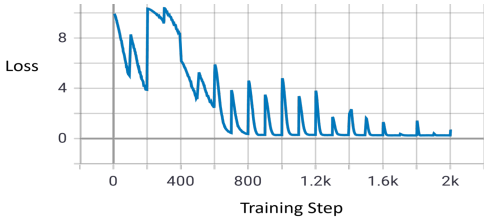


Fig. 7: Total loss graph at every training iteration during the backdoor injection to the ResNet18

causes spikes in the loss curve. We compare our method with baselines for both phases since our attack scenario includes offline and online phases. Recall that in the offline phase, the optimization takes place to find the vulnerable bit locations and generate a trigger pattern. First, we evaluate the modified models with the corresponding trigger patterns. Then, for each modified part of the weight parameters, we look for a matching target page location on the profiled memory, which constitutes the online phase. If multiple bits need to be flipped in the memory, we choose the one with the largest gradient value so that we get the maximum possible performance from the baselines. Finally, DRAM Match Rate r_{match} is calculated as explained in Section V-B. The experiment results are summarized in Table II.

BadNet and FT have no control over the N_{flip} since they do not introduce any constraints during the optimization. Therefore, in the offline phase, BadNet requires up to one and a half million bit flips to inject a backdoor successfully. Although FT modifies only the last layer while keeping the other layers constant, meaning fewer bit flips than BadNet, we observe that up to 8,667 bits have to be flipped. TBT has control on the number of modified parameters which enables partial control on the N_{flip} since the number of modified parameters limits the maximum value N_{flip} can get. Therefore, we select the results that reproduce their claimed performance in the original work [37] without modifying too many weight parameters and increasing the N_{flip} too much, and thus, decreasing r_{match} further. In the offline phase, TBT finds a much smaller number of bits compared to BadNet and FT due to the limit on the modified parameters. Our experiments show that the CFT+BR method successfully injects a backdoor into ResNet20 model with **91.24%** TA and **94.62%** ASR by flipping only **10 bits** out of 2.2 million bits in the DRAM. In ResNet32 and ResNet18, CFT+BR achieves 91.46% and 95.26% ASR, respectively, with a maximum of 1.66% degradation in the TA. We observe that N_{flip} values in BadNet and FT depend heavily on mode size. As the total number of bits increases, they require more bit-flips to achieve similar performance. On the other hand, we do not observe a significant dependence on the model size in TBT, CFT, and CFT+BR methods in terms of N_{flip}, TA, and ASR. In BadNet, FT, and TBT, the bit flips are concentrated within the same pages. Especially FT and TBT targets on the last layer of the DNN models. Since the last layer of the Resnet20, ResNet32, and ResNet18 models occupy only one

TABLE II: Comparison of our methods CFT, CFT+BR with the baseline methods BadNet, FT, and TBT on CIFAR10 [53] with ResNet-20/32/18, and ImageNet [54] with ResNet-34/50. Our proposed CFT+BR results are written in bold. Note that the percentage of the backdoor parameter bits ($\Delta\theta$) that are actually flippable, r_{match}, must be near 100% for a viable backdoor injection attack using Rowhammer.

			Offline Phase			Online Phase			
Dataset	Net	Method	N_{flip}	TA(%)	ASR(%)	N_{flip}	TA(%)	ASR(%)	r_{match}(%)
CIFAR10	ResNet20 Acc: 91.78% #Bits: 2.2M #Pages: 69	BadNet	172,891	86.96	99.98	33	91.76	2.63	0.02
		FT	2,238	84.36	97.10	1	91.72	2.90	0.04
		TBT	44	86.61	95.43	1	91.72	4.71	2.27
		CFT	22	90.09	99.55	5	91.79	14.40	22.73
		CFT+BR	**10**	**91.24**	**94.62**	**10**	**89.04**	**92.67**	**99.99**
	ResNet32 Acc: 92.62% #Bits: 3.7M #Pages: 116	BadNet	246,004	88.60	99.99	53	92.61	7.32	0.02
		FT	2318	81.87	90.59	1	92.65	8.57	0.04
		TBT	210	81.90	89.66	1	92.66	8.42	0.48
		CFT	39	90.25	98.75	10	92.41	20.22	25.64
		CFT+BR	**95**	**91.77**	**91.46**	**95**	**89.56**	**89.58**	**99.99**
	ResNet18 Acc: 93.10% #Bits: 88M #Pages: 2750	BadNet	1,493,301	87.61	99.88	416	93.06	12.45	0.03
		FT	8,667	88.80	95.34	1	92.20	34.16	0.01
		TBT	95	82.87	88.82	1	92.60	48.12	1.05
		CFT	42	92.39	99.90	11	91.52	0.36	26.19
		CFT+BR	**99**	**92.95**	**95.26**	**99**	**90.71**	**93.30**	**99.99**
ImageNet	ResNet34 Acc: 73.31% #Bits: 172M #Pages: 5375	BadNet	441,047	70.81	99.73	100	70.39	0.009	0.02
		FT	54,726	68.30	99.14	11	70.95	0.18	0.02
		TBT	553	72.69	99.86	1	70.97	0.05	0.18
		CFT	1509	70.25	99.76	388	69.93	0.10	25.71
		CFT+BR	**1463**	**70.28**	**72.92**	**1463**	**68.59**	**71.42**	**99.99**
	ResNet50 Acc: 76.13% #Bits: 184M #Pages: 5750	BadNet	359,516	73.98	99.11	129	66.43	0.05	0.04
		FT	93,778	68.43	96.52	12	73.77	0.09	0.01
		TBT	543	75.60	99.98	1	73.78	0.10	0.18
		CFT	1562	70.58	99.99	391	66.71	4.92	25.03
		CFT+BR	**1475**	**70.64**	**98.22**	**1475**	**68.94**	**96.20**	**99.99**

memory page in DRAM, the bit-flip locations found in the offline phase of FT and TBT reside within a single page. For instance, 210 bit-flips found by TBT on ResNet32 are all on the same page. However, as we mention in Section IV-A2, only the pages with one targeted bit location can be found in DRAM in practice. Therefore, we choose the bit flip with the largest gradient in a memory page and keep it modified and return the other parameters to their original values. Finally, we evaluate their performance on the test data set. In the ResNet20 and ResNet32 models, we observe that the ASR of BadNet, FT, and TBT drops down below 10% while the Test Accuracy values increase back to their original values. We claim that the significant decrease in ASR values can be explained by the diffusion effect of optimizing the parameters in an unconstrained way. When the attack is implemented on DRAM using Rowhammer, r_{match} values of BadNet, FT, and TBT are lower than 3% for every DNN model. In CFT, r_{match} is relatively higher than the other baseline methods since it modifies only one parameter in a page. However, it does not put a constraint on the number of bit flips within a byte during the optimization. Therefore, the attack performance degrades drastically in practice. In all experiments, CFT+BR has 99.9% r_{match} since it already considers the bit locations that can be flipped during the attack. Since the bit flips are sparse across different memory pages in CFT+BR, **100%** of the bit flips can actually be flipped. A small number of bits may be flipped in random locations, but it does not affect the performance of the attack significantly. We show that lower r_{match} values lead to low ASR in backdoor injection attacks using Rowhammer.

E. ImageNet Experiments

We also compare our method with the baseline methods on models trained on the ImageNet ILSVRC2012 Development Kit [54] data set, which consists of 1000 classes of visual objects. We used pre-trained ResNet34 and ResNet50 from the model zoo [52] as the target models. ResNet34 and ResNet50 include 172 million and 184 million bits, respectively. Note that both the model and data set sizes are significantly larger compared to our CIFAR-10 experiments. As the TA and ASR, we use top-1 accuracy results. The results are summarized in Table II. The same comparison methods we apply in CIFAR-10 are valid in ImageNet experiments as well.

In the offline phase of the attacks, we observe that each method shows a different response to the increase in the model and data set sizes. For instance, BadNet and FT require more than 350K and 50K, respectively. Compared to CIFAR-10 models, BadNet is not affected significantly. However, N_{flip} for FT becomes 17 times larger on average on the ImageNet models. TBT locates around 550 N_{flip} on the ResNet34 and ResNet50 models in the offline phase, which is 5 times larger on average than the CIFAR-10 experiments. CFT and CFT+BR locate around 1500 N_{flip} on the ResNet34 and ResNet50 models in the offline phase, meaning 45 times and 22 times larger for CFT and CFT+BR, respectively.

In the online phase, we observe that none of the baseline methods has a significant attack performance. For instance, in the BadNet method, although the model sizes increase 5.5 times, the number of modified pages increases only 1.5 times on average. Similarly, TBT modifies only one page in the last layer of the ResNet34 and ResNet50 models, even though the last layers of the models have more than 10 pages. This clearly shows that as the model size increases, the density of bit flips required by the baseline models increases, meaning the attack tends to focus on certain regions instead of uniformly distributing the bit flips. The high density of the bit flips leads to r_{match} rates as low as 0.02%. Although FT modifies most of the pages in the last layer, the fact that the bit locations are not optimized at the beginning causes vanishing ASR. Overall, we observe that the claimed ASRs can be achieved only when r_{match} is large enough. Although CFT achieves much larger r_{match} values than the other baseline methods, lacking *Bit Reduction* makes the attack focus on multiple bit flips within 8-bit parameters, which, in return, causes lower than 5% ASR on the models trained with ImageNet data set. In contrast, CFT+BR can inject the backdoor to ResNet models with up to 96.2% ASR and a maximum of 7.2% degradation in the TA, which makes it the best-performing backdoor injection attack compared to the baseline methods. These results show that our approach generalizes well to larger data sets and models. Note that although N_{flip} increases as the model gets larger in CFT+BR, it is still possible to flip these bits with 99.99% r_{match} due to the sparse distribution.

F. Generalization to Other DNN Architectures

We experiment on other DNN architectures, such as VGG11, VGG16, to show that our attack generalizes. We show that CFT+BR can successfully locate vulnerable bits and achieves over 90% Attack Success Rate in VGG architectures. The results are summarized in Table III.

TABLE III: CFT+BR experiment results on VGG architectures

Model	Base Acc	TA [%]	ASR [%]	N_{flip}
VGG11	92.35	92.70	100	30
VGG16	92.68	92.57	90.85	100

VI. Potential Countermeasures

We analyze some of the prominent countermeasures proposed for mitigating bit-flip attacks against DNN models.

A. Prevention-Based Countermeasures

Binarization-Aware Training [56][2] is a method that uses *Binarized Neural Networks (BNNs)* [57], [58] to increase the resistance of DNNs against the bit flip attacks. This method significantly reduces the network size. For instance, a binarized ResNet-32 model occupies only 65 pages in the memory. Although 65 bit flips are not enough to inject a backdoor using Rowhammer, N_{flip} cannot be larger than the number of pages occupied by the model. Therefore, our experiments show that using BNNs is an effective defense against our attack since it aggressively decreases the size of the network and, consequently, the maximum N_{flip}. However, reducing the model size causes accuracy degradation as a performance overhead. Note that BNNs may still be vulnerable to other fault attacks which do not require the same physical constraints, such as sparse faulty bit locations.

Piecewise Weight Clustering (PWC) [56] is a relaxation of BNNs. With PWC, an additional penalty term is introduced to the inference loss function, which forces model weight distribution to form two clusters. We experiment with our attack against a ResNet32 model trained with PWC penalty term in the loss function. We observe a strengthened trade-off between the TA and ASR during the optimization.For instance, the ASR drops down to 43.42% when TA is 89.66% with 112 N_{flips}. On the other hand, our attack achieves 98.49% ASR while degrading the TA down to 9.9% with the same N_{flips}. The results show that training the model with PWC does not protect against accuracy degradation and even targeted misclassification attacks. However, it makes it harder to inject stealthy backdoors.

B. Detection-Based Countermeasures

Possible defense techniques focusing on detecting the attacks on the model weights [59]–[62] come with an overhead because they need to be deployed together with the model into the machine learning product.

DeepDyve [59] is a dynamic verification method that uses a checker model along with the original model for mitigating the transient faults in the inference. It assumes both models predict the same results for the same inputs most of the time. When the results of the two models are the same, the result is accepted immediately. However, if the results are different, the inference is repeated, and the second result from the original model is accepted. DeepDyve assumes the fault in the model is transient and does not appear in consecutive queries. However, the bit flips introduced by Rowhammer stay in the memory until being reloaded from the disk. Since the transient assumption does not hold, even if a checker model raises an alarm and repeats the inference, the new inference is made by the backdoor-injected model and will not be detected.

Weight Encoding [60] proposes additional matrix multiplication and weight extraction. Thus, this method can detect only the topmost sensitive layers in the network to keep the overhead low. However, our attack can target all layers to inject a backdoor. Therefore, the spatial locality assumption does not hold with our attack. Using the overhead numbers in [60] for ResNet-34, we estimate the time and storage overhead against our attack. Since the time complexity of weight encoding $d_j = r(y_j), y_j = \phi(\sum_{i=0}^{N-1} B_i \cdot K_{ij})$ is $O(N^2)$, where B is Z^N, and K is R^{NxM}, the estimated execution time overhead of the method is 834.27 seconds. Since the storage complexity of the Weight Encoding is linear, the storage cost for ResNet34 is estimated as $(0.141/8192) \times 21779648 = 374.86MB$,

[2]Binarization-Aware Training and Piecewise Weight Clustering implementations are taken from https://github.com/elliothe/BFA.

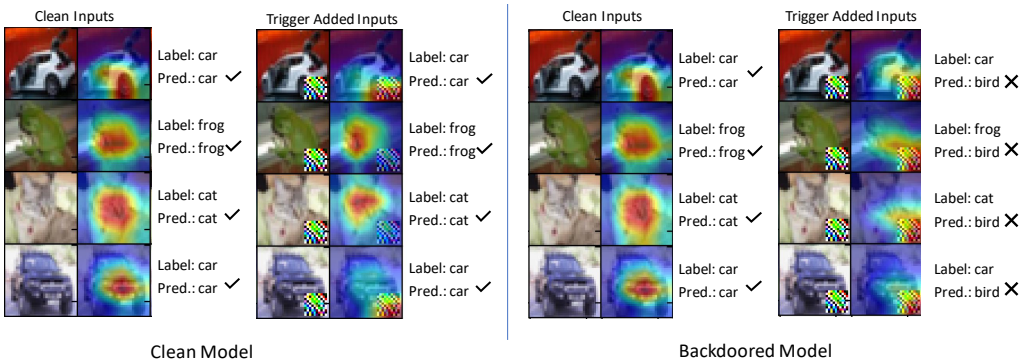


Fig. 8: The change in GradCAM [55] heatmaps that belong to ResNet18 before the attack (left) and after the attack (right). The focus of the model shifts through the trigger pattern if it is backdoored.

which is 446% storage overhead, showing that the proposed method is not scalable.

RADAR *[61]* is a checksum-based detection method during inference. It divides the weights into groups and gets the checksum of the most significant bits of parameters in each group. The original checksum values of the parameters are stored along with the model and are validated with the original signatures at every inference time. The optimization constraints can be further increased to avoid flipping the MSB of the weight parameters in our attack, which can bypass the detection. Assuming linear time complexity, time overhead goes up to 40.11% for full-size bit protection in ResNet20.

SentiNet [62] filters the adversarial inputs using GradCAM heatmaps [55]. We use the GradCAM implementation from [63] to analyze the output of four sample images that are labeled as *car, frog, cat* and *car* respectively (See Figure 8.). Before the attack, the model correctly classifies all images with or without the trigger pattern. If the trigger pattern does not overlap with the major features in the image, e.g. *frog* and *cat*, the main focus of the model stays on the object. However, if the trigger pattern overlaps with the main features, e.g. the wheel of the *car*, the focus is shifted towards the trigger pattern. After the attack, regardless of the trigger and object overlap, the focus of the model shifts towards the trigger pattern, and the model misclassifies all images to the target class, *bird*. Therefore, although a GradCAM-based approach can possibly filter the adversarial inputs, it will also produce false positives even if the model is clean and works correctly.

C. Recovery-based Countermeasures

Weight Reconstruction: Li et al. [64] propose *Weight Reconstruction*[3] to recover the clean network after a bit flip attack occurs. *Weight Reconstruction* aims to recover from an accuracy degradation caused by the attack. After a bit flip occurs in a weight parameter, the effect of the change is distributed onto other parameters to reduce the overall effect on the model performance. We experiment with our CFT+BR attack against a ResNet32 defended by Weight Reconstruction to evaluate the effectiveness of the proposed defense method. We applied our attack in two different scenarios. In the first scenario, the attacker is not aware that the model is defended by Weight Reconstruction and applies the offline phase of the attack as described in Section IV-A3. As a baseline, our attack achieves 91.46% ASR and 97.77% TA by flipping 95 bits in the memory. After applying Weight Reconstruction, we observed that ASR and TA become 32.89% and 91.02, respectively. In the second scenario, the attacker is aware that the model is defended by Weight Reconstruction and applies the offline phase of the attack against a model with Weight Reconstruction. However, if the attacker is aware of the defense and applies CFT+BR on a defended model, our attack successfully bypasses Weight Reconstruction by achieving 94.04% ASR and 89.51% TA. Therefore, the Weight Reconstruction approach does not protect the models when the attacker knows the applied defense.

[3]Weight Reconstruction implementation is taken from https://github.com/zlijingtao/DAC20_reconstruction.

VII. Related Works

a) Rowhammer Attacks on DNNs: We compare our work with Terminal Brain Damage [4] and Deephammer [5] in terms of the following factors:

Attacker's Objectives: The main difference between our work and previous works is the goal of the attack. In both [4] and [5], the attacker's objective is to degrade the inference accuracy of the model on legitimate inputs and cause a denial of service. In contrast, our attack objective in this work is to keep the inference accuracy for legitimate inputs the same and misclassify all trigger-added inputs to a target class in stealth by using a unified objective function given in Equation 3.

Assumptions: All [4], [5], and our work assume the attack takes place in a cloud environment where the model is loaded into system's shared memory and stays unchanged. Unlike [5], we do not assume the availability of huge page configuration to bypass virtual to physical translation.

Attacker Capabilities: Same as our attack, [5] and [4] assume the attacker knows the model architecture and parameters. [4] also considers black-box setting with random bit flips. Since our attack objective is more sophisticated, our attack is not applicable in a black-box setting.

Attack Time: [5] configures the hammering time for each row as 190ms. Since [4] only simulates the attack, they assume it is 200ms in the calculations. In our setup, it takes 800ms to hammer one row using a 15-sided pattern during the profiling phase and 400ms using a 7-sided pattern during the online phase. Note that previous works consider only double-sided Rowhammer, which takes less time but is not effective on DDR4 chips with TRR mitigation. Total online attack time varies between different models and can be estimated by multiplying the hammering time by N_{flip}.

Stealthiness/Detectability: Due to the difference in the attack objectives, the stealth of the attacks is also different. For instance, Test Accuracy after the attack on VGG16 is given as around 10% in both [4] and DeepHammer. However, we can preserve the Test Accuracy at over 92% after our attack while being able to misclassify over 90% of all instances with an attacker-generated trigger pattern. Since we can preserve the Test Accuracy close to the base accuracy of the models, our attack is stealthy.

Comparison of Accuracy Degradation: Although the goal of backdoor injection is not accuracy degradation, the resulting degradation on trigger-added inputs is comparable to [4] and [5]. In VGG16 trained on CIFAR10, when we add trigger pattern to all images, we see the accuracy of the model to be 18% (an 80% relative accuracy degradation from baseline). Alternatively, [5] and [4] claim relative accuracy degradations of VGG16 to be 88% and 90%, respectively (after the attack the models only produce a correct output 10% of the time).

b) Accuracy Degradation Attacks: Bit-Flip Attack [65] degrades the accuracy of DNN models to random guess using a chain of bit flips. Targeted Bit-Flip Attack [66] is shown to be capable of misclassifying the samples from single or multiple classes to a target class on quantized DNN models. Although these works show that DNN model performance can be damaged permanently by flipping a limited number of bits in the weight parameters, these attacks do not make use of an attacker-controlled backdoor trigger. Therefore, they have very limited control over stealthiness. A binary integer programming-based approach was proposed by Bai et al. [7] to find the minimum number of bit flips required to make the model misclassify a single image sample into a targeted class.

c) ML Backdoor Attacks: Garg et al. [67] observed that adversarial perturbations on the weight space of the trained models could potentially inject Backdoor, but it requires either social engineering or full privileged access to replace the target model with the backdoored model. Recently, [37] and [33] showed that backdoor attacks could be implemented by changing only a small number of weight parameters. However, both of the works assume any bit location in the memory can be flipped, which is not practical. Therefore, the practicality of software-based backdoor injection attacks during the inference phase is still an open question due to the practical constraints that have been overlooked in previous works.

VIII. Discussion

Effect of Huge Pages: We assume huge pages are not available since they give an advantage for finding contiguous memory in physical address space. Even though the target model uses huge pages, the memory controller would still fragment the huge page into 8 KB rows in DRAM due to the fixed row size. Also, each chunk is mapped into different banks in order to increase parallel access. For example, if there are 64 banks in the system, a 2 MB huge page would be fragmented into 64 chunks and 4 neighbor rows in the DRAM. Although this may hurt the n-sided Rowhammer pattern, it would still be possible to sandwich each chunk and do Rowhammer. Note that, in memory systems with multiple DIMMs, and ranks, the number of banks also increases, which would decrease the size of the chunks down to a single row. In that case, a regular double-sided or n-sided Rowhammer attack would still work. Since an attacker can choose to profile 4 KB pages in DRAM, finding 512-bit flips in 2 MB would still be practical.

Application on Other Security Critical Tasks The proposed attack method is a generic approach and agnostic to the downstream tasks. Therefore, it would work on models used in other safety-critical tasks, such as voice recognition applications.

IX. Conclusion

We analyzed the viability of a real-world DNN backdoor injection attack. Our backdoor attack scenario applies to deployed models by flipping a few bits in memory assisted by the Rowhammer. Our initial analysis performed on hardware showed that earlier proposals fall short in assuming a realistic fault injection model. We devised a new backdoor injection attack method that adopts a combination of trigger pattern generation and sparse and uniform weight optimization. In contrast to earlier proposals, our technique uses all layers and combines trigger pattern generation, target neuron selection, and fine-tuning model parameter weights in the same training loop. Since our approach targets the weight parameters uniformly, it is guaranteed that no more than one bit in a memory page is flipped. Further, we introduced new metrics to capture a realistic fault injection model. This new approach achieves a viable solution to target real-life deployments: on CIFAR10 (ResNet 18, 20, 32 models) and ImageNet (Resnet34 and 50 models) on real hardware by running the Rowhammer attack achieving Test Accuracy and Attack Success Rates as high as 92.95% and 95.26%, respectively. We also showed that our attack works on other architectures, such as VGG11 and VGG16. Finally, we evaluated the prominent defense techniques against our backdoor injection attack. We concluded that the proposed countermeasures are either not effective or introduce significant overhead in terms of time and storage.

Acknowledgements

We thank our anonymous reviewers for their insightful feedback. This work is supported by the National Science Foundation, under grants CNS-1814406, CNS-2026913, and CCF-2006738, and by the U.S. Department of State, Bureau of Educational and Cultural Affair's Fulbright Program.

REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[3] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

[4] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 497–514.

[5] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 1463–1480.

[6] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 131–138.

[7] J. Bai, B. Wu, Y. Zhang, Y. Li, Z. Li, and S.-T. Xia, "Targeted attack against deep neural networks via flipping limited weight bits," *arXiv preprint arXiv:2102.10496*, 2021.

[8] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[9] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 638–651.

[10] L. Orosa, A. G. Yaglikci, H. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim, and O. Mutlu, "A deeper look into rowhammer's sensitivities: Experimental analysis of real dram chips and implications on future attacks and defenses," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1182–1197.

[11] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack." in *USENIX Security symposium*, vol. 25, 2016, pp. 1–18.

[12] V. Van Der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1675–1689.

[13] K. Mus, S. Islam, and B. Sunar, "Quantumhammer: a practical hybrid attack on the luov signature scheme," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1071–1084.

[14] S. Islam, K. Mus, R. Singh, P. Schaumont, and B. Sunar, "Signature correction attack on dilithium signature scheme," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2022, pp. 647–663.

[15] M. Fahr Jr, H. Kippen, A. Kwong, T. Dang, J. Lichtinger, D. Dachman-Soled, D. Genkin, A. Nelson, R. Perlner, A. Yerukhimovich *et al.*, "When frodo flips: End-to-end key recovery on frodokem via rowhammer," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 979–993.

[16] K. Mus, Y. Doröz, M. C. Tol, K. Rahman, and B. Sunar, "Jolt: Recovering tls signing keys via rowhammer faults," *Cryptology ePrint Archive*, 2022.

[17] P. Frigo, E. Vannacc, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "Trrespass: Exploiting the many sides of target row refresh," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 747–762.

[18] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "Smash: Synchronized many-sided rowhammer attacks from javascript," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[19] P. Jattke, V. Van Der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable rowhammering in the frequency domain," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 716–734.

[20] H. Hassan, Y. C. Tugrul, J. S. Kim, V. Van der Veen, K. Razavi, and O. Mutlu, "Uncovering in-dram rowhammer protection mechanisms: A new methodology, custom rowhammer patterns, and implications," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1198–1213.

[21] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 55–71.

[22] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are we susceptible to rowhammer? an end-to-end methodology for cloud providers," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 712–728.

[23] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation," in *25th {USENIX} security symposium ({USENIX} security 16)*, 2016, pp. 19–35.

[24] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer attacks over the network and defenses," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 213–226.

[25] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice, and D. Gruss, "Nethammer: Inducing rowhammer faults through network requests," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 710–719.

[26] S. Qazi, Y. Kim, B. Boichat, E. Shui, and M. Nissler, "Introducing half-double: New hammering technique for dram rowhammer bug," https://github.com/google/hammer-kit, 2021.

[27] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 245–261.

[28] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[30] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *Proceedings of the 27th USENIX Conference on Security Symposium*, ser. SEC'18. USA: USENIX Association, 2018, p. 1615–1631.

[31] M. Shafieinejad, N. Lukas, J. Wang, X. Li, and F. Kerschbaum, "On the robustness of backdoor-based watermarking in deep neural networks," in *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, ser. IHamp;MMSec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 177–188. [Online]. Available: https://doi.org/10.1145/3437880.3460401

[32] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[33] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Proflip: Targeted trojan attack with progressive bit flips," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 7718–7727.

[34] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-5_Liu_paper.pdf

[35] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," *arXiv preprint arXiv:2005.03823*, 2020.

[36] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," *arXiv preprint arXiv:1806.05768*, 2018.

[37] A. S. Rakin, Z. He, and D. Fan, "Tbt: Targeted neural network attack with bit trojan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 198–13 207.

[38] H. Yu, K. Yang, T. Zhang, Y.-Y. Tsai, T.-Y. Ho, and Y. Jin, "Cloudleak: Large-scale deep learning models stealing through adversarial examples." in *NDSS*, 2020.

[39] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in

Proceedings of the 2017 ACM on Asia conference on computer and communications security, 2017, pp. 506–519.
[40] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 601–618.
[41] J. R. Correia-Silva, R. F. Berriel, C. Badue, A. F. de Souza, and T. Oliveira-Santos, "Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.
[42] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "Prada: protecting against dnn model stealing attacks," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 512–527.
[43] S. Islam, A. Moghimi, I. Bruhns, M. Krebbel, B. Gulmezoglu, T. Eisenbarth, and B. Sunar, "{SPOILER}: Speculative load hazards boost rowhammer and cache attacks," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 621–637.
[44] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "{DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks," in *25th USENIX security symposium (USENIX security 16)*, 2016, pp. 565–581.
[45] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating software mitigations against rowhammer: a surgical precision hammer," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 47–66.
[46] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.
[47] A. Chakraborty, S. Bhattacharya, S. Saha, and D. Mukhopadhyay, "Explframe: Exploiting page frame cache for fault analysis of block ciphers," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1303–1306.
[48] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 695–711.
[49] S. Migacz, "8-bit inference with TensorRT," *NVIDIA GPU Technology Conference*, 2017.
[50] NVIDIA, "Tensorrt documentation," https://docs.nvidia.com/deeplearning/tensorrt, 2021, accessed: 2021-05-25.
[51] Y. Idelbayev, "Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch," https://github.com/akamaster/pytorch_resnet_cifar10, 2019, accessed: 2021-05-26.
[52] Torchvision, https://pypi.org/project/torchvision/, 2021, accessed: 2021-05-26.
[53] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
[54] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
[55] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.
[56] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, "Defending and harnessing the bit-flip based adversarial weight attack," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 095–14 103.
[57] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Advances in neural information processing systems*, vol. 29, 2016.
[58] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.
[59] Y. Li, M. Li, B. Luo, Y. Tian, and Q. Xu, "Deepdyve: Dynamic verification for deep neural networks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 101–112.
[60] Q. Liu, W. Wen, and Y. Wang, "Concurrent weight encoding-based detection for bit-flip attack on neural network accelerators," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–8.
[61] J. Li, A. S. Rakin, Z. He, D. Fan, and C. Chakrabarti, "Radar: Runtime adversarial weight attack detection and accuracy recovery," *arXiv preprint arXiv:2101.08254*, 2021.
[62] E. Chou, F. Tramer, and G. Pellegrino, "Sentinet: Detecting localized universal attacks against deep learning systems," in *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2020, pp. 48–54.
[63] J. Gildenblat and contributors, "Pytorch library for cam methods," https://github.com/jacobgil/pytorch-grad-cam, 2021.
[64] J. Li, A. S. Rakin, Y. Xiong, L. Chang, Z. He, D. Fan, and C. Chakrabarti, "Defending bit-flip attack through dnn weight reconstruction," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
[65] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1211–1220.
[66] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, "T-bfa: Targeted bit-flip adversarial weight attack," *arXiv preprint arXiv:2007.12336*, 2020.
[67] S. Garg, A. Kumar, V. Goel, and Y. Liang, "Can adversarial weight perturbations inject neural backdoors?" *CoRR*, vol. abs/2008.01761, 2020. [Online]. Available: https://arxiv.org/abs/2008.01761
[68] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

APPENDIX

A. Probability Analysis

We further analyze Equation 2 with the numbers calculated in the Table I.

First, we calculate the probability of finding a target page t for each N values and three different $k+l$ values. Note that $k+l$ is the number of bit offsets within a page. Figure 9 shows that, for 1 bit per page, 2200 pages are enough to achieve 99.99% accuracy for the DDR4 DRAM K1 listed on Table I. For 2 and 3 bits per page, the same number of pages give 2% and 0.006% probability, respectively.

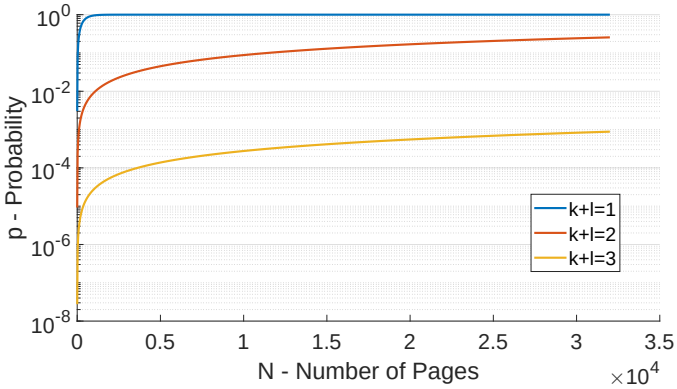


Fig. 9: Probability of finding a page among N pages for different k+l values. k+l states the number of targeted bit offsets in a page.

Second, given a page offset to flip, we calculate the probability of finding such a target page for each N values and different DRAM chips. The results on Figure 10 suggest that given enough number of pages, N, the probability of finding a target page is close to 1 for even the least flippy DRAM devices.

B. Finding Contiguous Memory

Virtual to physical address mappings are stored in *pagemap* file in Linux OSs and it requires root privileges to access these

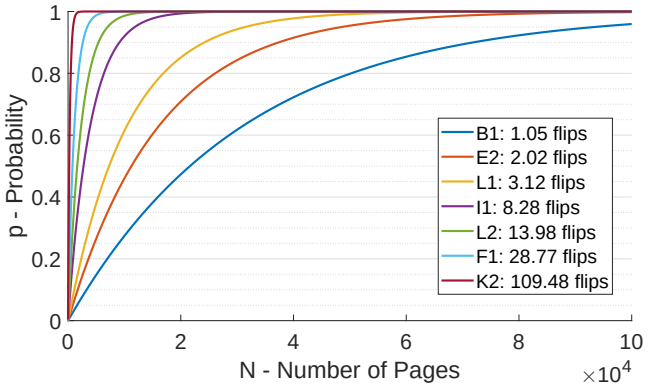


Fig. 10: Probability of finding a page among N pages for different DRAM chips.

translations. Using the SPOILER tool [43], we can reliably leak the information about the last 8 bits in physical addresses after the page offset bits. This allows us to find contiguous memory chunks in physical address space.

SPOILER works by taking advantage of a performance optimization in Intel processors where loads are executed speculatively before stores, resulting in a timing side channel if the load has dependencies based on the same partial address information as a store. SPOILER uses the differences in time between loads and stores to extrapolate which virtual addresses within an array are contiguous physically. Our SPOILER implementation performs the timing measurements 100 times per page, removing outliers and taking the average read time.

In Figure 11, the page numbers with the peaks in the y-axis are contiguous pages physical address space.

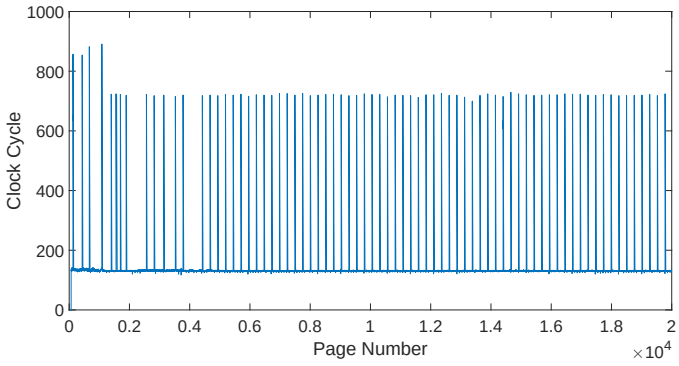


Fig. 11: Timing peaks on virtual addresses detected by SPOILER [43] attack. Virtual addresses on the peaks are contiguous on physical address space.

C. Finding Neighbor Rows in DRAM

After translating the address from virtual to physical address space, there is another translation which maps the physical addresses to DRAM banks, rows and columns. To be able to successfully realize a Rowhammer attack, we need physical addresses that are mapped to the same bank in the DRAM, and thus, neighbor rows. We use row buffer conflict side-channel [44] to detect which two addresses are in the same bank. Figure 12 shows that about one sixteenth of the address give larger access time, meaning they are in the same banks and neighbor rows.

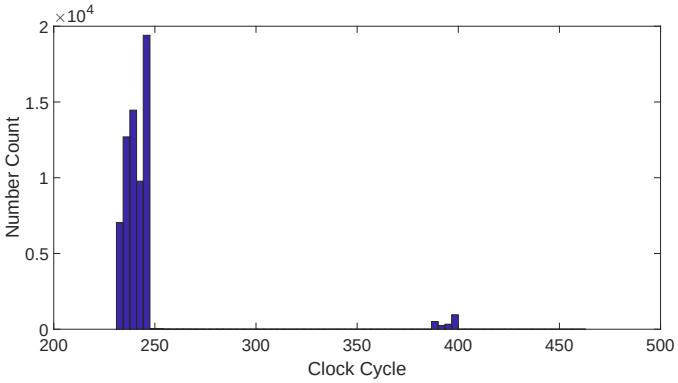


Fig. 12: Access time distribution of previously found contiguous physical addresses. Accessing physical addresses that are mapped to the same DRAM bank takes around 400 clock cycles.

D. Restoring the Modified Parameters

In order to show the importance of putting constraints on the optimization of target neurons, first, we fine-tune all parameters in a ResNet18 model using clean and adversarial examples without putting any constraints. Since the training approach is aligned with the work in [32], we refer to this method as BadNet. Then, starting from the weight parameters with the lowest gradient values, we restore a part of the parameters to their original values at the end. Table IV shows the change in the attack performance a part of the weight parameters are restored into their original values. For instance, BadNet reaches 99.88% Attack Success Rate and 87.61% Test Accuracy after the unconstrained fine-tuning. Then, when we restore only 1% of the weights (i.e. 99% of the weights remain modified), we observe that Attack Success Rate drops down to 76.11% while the Test Accuracy is slightly increasing. Even if we keep 50% of the parameters (44 million bits) modified, limiting the number of modified parameters at the end of fine-tuning achieves only 34.15% Attack Success Rate, whereas we reach 92.95% test accuracy and 95.26% attack success rate with only 99 bit-flips using CFT+BR. Therefore, we claim that fine-tuning without any constraints distributes the knowledge of backdoor to all parameters and the parameter limit that is applied at the end degrades the backdoor success rate drastically. This result pushes us towards putting constraints on fine-tuning.

E. Comparison of Found Bit Flips

Figure 13 illustrates the sparsity of found bit flip locations by our method (CFT+BR) and TBT. Note that the bit flips found by TBT (red) are localized in only one page. However, the bit flips located by CFT+BR are sparsely distributed over

TABLE IV: BadNet reaches reasonable attack success rate (ASR) and test accuracy (TA) only when more than 90% or parameters are changed. Limiting the percentage of modifications after fine-tuning decreases the attack performance.

Modification(%)	TA(%)	ASR(%)
100	87.61	99.88
99	89.79	76.11
90	90.92	61.04
80	91.67	51.22
70	92.01	43.79
50	92.41	34.15

the weight file which makes them actually flippable in the online phase.

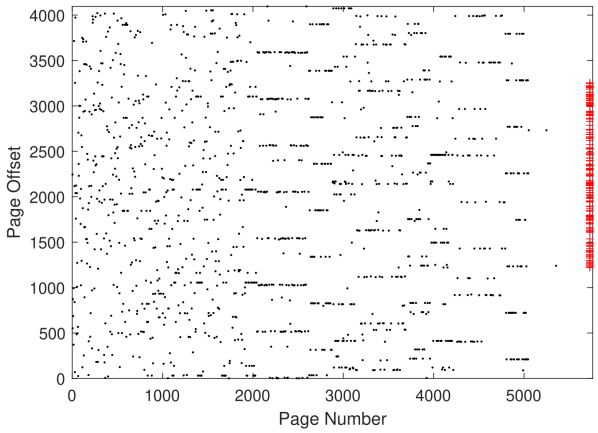


Fig. 13: The comparison of vulnerable bit locations found by CFT+BR (black) and TBT (red) on ResNet50 weight file.

F. Negative Result - Plundervolt Attack

In this experiment, we try to use another software-based faulting mechanism, namely Plundervolt [68], to inject faults during the inference phase of a DNN model. Differently from the Rowhammer attack, the Plundervolt attack utilizes undervolting the CPU beyond the optimal operation limits using the MSR interface to cause faulty results in the multiplication results. Since the computational graphs of DNN models have many multiplication operations, Plundervolt can be a potential threat to DNN inference as well. We first run the Plundervolt PoC code to verify undervolting can fault the multiplication operations and get the frequency-voltage pair where we can reliably produce faults. Then, we experimented with DNN models with floating-point weight parameters and undervolted the CPU to the determined frequency-voltage pair. We did not observe any faults in the multiplication results when the operands are floating points. We also experimented undervolting while an n-bit quantized DNN model is operating. However, we did not see any faults in the DNN model. We claim that the reason why the multiplication results are not affected by undervolting is the operand values are limited to $2^n - 1$ which is 255 in 8-bit quantized DNN models. We observed that when the second operand of multiplication is smaller than `0xFFFF`, undervolting does not introduce any bit flips in the multiplication result which is consistent with the observations in the original Plundervolt work [68].

We also experimented with the matrix multiplication implementations of the PyTorch library. We observe that `torch.matmul` function produces faulty results only when the following three conditions are met. First, the second operand must be larger than `0xFFFF`. Second, the size of operands must be 1-by-1. Finally, the multiplication operation must be run in a while loop keeping operands constant.

Hence, we conclude that injecting backdoors to the DNN models or degrading the accuracy of them using Plundervolt attack is not practical.