ModelShield: A Generic and Portable Framework Extension for Defending Bit-Flip based Adversarial Weight Attacks

Yanan Guo
University of Pittsburgh
Pittsburgh, PA, USA
yag45@pitt.edu

Liang Liu
University of Pittsburgh
Pittsburgh, PA, USA
lil125@pitt.edu

Yueqiang Cheng
NIO
San Jose, CA, USA
strongerwill@gmail.com

Youtao Zhang
University of Pittsburgh
Pittsburgh, PA, USA
zhangyt@cs.pitt.edu

Jun Yang
University of Pittsburgh
Pittsburgh, PA, USA
juy9@pitt.edu

2021 IEEE 39th International Conference on Computer Design (ICCD) | 978-1-6654-3219-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCD53106.2021.00090

***Abstract*—Bit-flip attack (BFA) has become one of the most serious threats to Deep Neural Network (DNN) security. By utilizing Rowhammer to flip the bits of DNN weights stored in memory, the attacker can turn a functional DNN into a random output generator. In this work, we propose ModelShield, a defense mechanism against BFA, based on protecting the integrity of weights using hash verification. ModelShield performs real-time integrity verification on DNN weights. Since this can slow down a DNN inference by up to 7×, we further propose two optimizations for ModelShield. We implement ModelShield as a lightweight software extension that can be easily installed into popular DNN frameworks. We test both the security and performance of ModelShield, and the results show that it can effectively defend BFA with less than 2% performance overhead.**

***Index Terms*—GPU, Machine Learning, Rowhammer**

I. Introduction

Recently, deep neural networks (DNNs) based machine learning (ML) algorithms have shown their great potential in multiple fields [1], [5]. Instead of investing in their own DNN models, a lot of companies and personal users prefer to use Machine-Learning-as-a-Service (MLaaS) cloud server platforms. MLaaS significantly reduces the effort and cost of developing and maintaining ML applications. However, there is usually more than one application running on the server at the same time, and the hardware resources on the server are shared between the ML application and other co-running applications. With this setup, the internal DNN model of an ML application, which is typically stored in the main memory of the server, can be modified by the co-located malicious application (the attacker) indirectly, using Rowhammer [6].

Rowhammer is a security exploit that alters the 1-bit data stored in a memory cell by repeatedly accessing cells in its neighboring rows. Previous works have shown that Rowhammer can be utilized to flip the bits of DNN weights and significantly reduce the inference accuracy or even make the DNN a random output generator [18]. Recently, a very efficient attack named Bit-Flip based Attack (BFA) [12] was proposed where the attacker efficiently identifies and flips only a small number of *most vulnerable bits* in a DNN model. Several defense mechanisms against BFA have been proposed, including binarization-aware training [4], and weight reconstruction [9]. However, these methods are only making DNN models more robust against BFA instead of strictly protecting them from BFA.

Challenge. One way to prevent BFA is to protect the integrity of DNN weights using hash verification. However, a naive hash verification design could cause both security and performance problems. First, using unkeyed hashes for verification may not protect the model: the attacker could modify the hashes stored in memory as well to ensure they match the modified weights. Thus, one may consider using a keyed hash and keeping the key on chip instead of storing it in memory. However, this requires hardware modification, which significantly complicates the protection. Second, to avoid the time-of-check to time-of-use problem, we need to perform real-time hash verifications. Unfortunately, this can slow down the inference by up to 7×, making it an impractical protection for MLaaS platforms.

In this work, we propose ModelShield, a lightweight defense mechanism that can *strictly* prevent BFA by protecting the integrity of DNN weights. ModelShield is implemented as a software extension that can be simply added into modern frameworks (e.g., Pytorch [11]). We overcome the mentioned security and performance challenges by thoroughly analyzing the strength of BFA and optimizing performance while maintaining security based on the analysis. Specifically, we first prove that a BFA attacker is not able to precisely modify the hash to make it match the corrupted weights, which gives us a chance to protect weights without a keyed hash and hardware modification. Second, we summarize the critical features of hashes for defending BFA and explore hashes that not only have these features but also good performance. Third, to further reduce the overhead, we build a software hash tree and find the tree structure that provides optimal performance.

Since most DNN inferences are performed on GPUs, we implement ModelShield in a CUDA kernel and build a script to link this kernel with modern frameworks. With ModelShield, programmers can verify the integrity of weights in one-line python code. We test ModelShield with popular DNN models, and the experimental results show that using ModelShield can successfully protect DNN models from BFA with less than 2% performance overhead and zero accuracy degradation.

II. Background and Prior Works

A. Rowhammer Attack

Modern DRAM-based memory chips consist of a two-dimensional array of cells. Each cell stores 1-bit information, represented by the charge of the capacitor in the cell. In 2014, it was found out that current DRAMs are vulnerable

978-1-6654-3219-1/21/$31.00 ©2021 IEEE
DOI 10.1109/ICCD53106.2021.00090

to disturbance errors induced by adjacent row activation [6]: activating a row in a DRAM bank can cause a little disturbance in its neighboring row; with frequently activating/accessing the same row (i.e. hammering the same row), the disturbance on the neighboring row will accumulate and eventually become significant enough to flip the stored bits in this row, before it gets refreshed. With Rowhammer, attackers are able to change the memory data of a co-located application without even accessing it directly. Very recently, Fan et al. discovered that Rowhammer attacks can be used to modify the weights of a functional DNN and make it a random output generator [18].

B. Bit-Flip based Adversarial Weight Attack

Adversarial weight attack is one of the main challenges on DNN security: even small changes to weights can lead to significant differences in inference accuracy [12]. The bit-flip based adversarial weight attack (bit-flip attack for short) is one of such adversarial attacks. This attack performs weight modification by flipping the bits of DNN weights stored in memory, utilizing well-developed Rowhammer tools. In 2019, Adnan et al. proposed a very efficient Bit-Flip Attack [12] (aka. BFA). Using their proposed Progressive Bit Search (PBS) algorithm, a BFA attacker can compromise a quantized DNN model with only several bit-flips.

C. Previous Defenses

Binarization-aware training: Zhezhi et al. made an important observation that BFA is prone to identify vulnerable bits in close-to-zero weights, and modify them to be large values [4]. Based on this observation, they proposed to use binarization-aware training to defend against BFA. In this training method, weights that are originally in floating point are converted to be represented by a binary-based format, and thus can be trained to stay *far from zero* to make the model more robust against BFA.

Weight reconstruction: When the attacker flips a bit in a weight, it induces a change of Δw on the target weight, which will affect the loss. Thus, to defend BFA, Jingtao et al. proposed weight reconstruction method which can reduce the Δw caused by a bit-flip and thus the overall increase in loss [9]. This weight reconstruction consists of three steps: 1) averaging, which spreads the effect of $|\Delta w|$ on a group of size G; 2) quantization, which cancels the effect of this $|\Delta w/G|$ change on the quantized mean; 3) clipping, which restricts all the weights to a small range near the quantized mean.

Limitations of previous works: The discussed previous defenses focus on modifying DNN models themselves to make them more robust against BFA. But this kind of method can only weaken BFA, rather than preventing it. As an example, without any defense mechanism, the BFA attacker only needs to flip the most significant bits (MSBs) of about 28 weights in a ResNet-20 model to make it generate random outputs on CIFAR-10. In contrast, with binarization-aware training, the attacker now needs to flip at least 500 MSBs. However, an attacker can easily find 500 weights with vulnerable MSBs, which is less than 0.2% of the total weights.

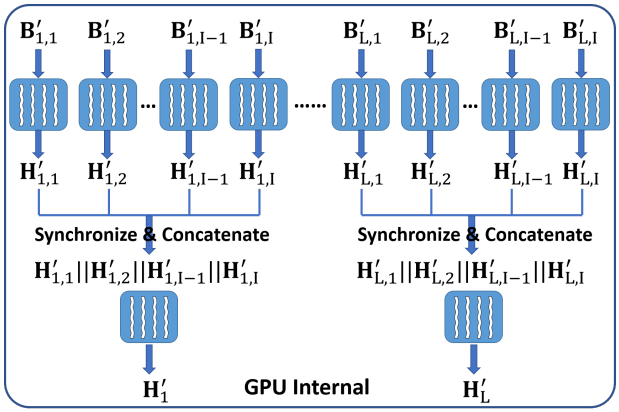

Fig. 1: Parallelizing hash verification in GPU by building a 2-level hash tree; each wavy line represents a GPU thread.

III. The Design of ModelShield

A. Design Overview

We propose ModelShield which can defend BFA by verifying the integrity of weights every time after the DNN inference is finished but before the result is sent back to the user. Specifically, we use a cryptographic non-keyed hash to pre-calculate the hash value over weights in each layer of the DNN model, and store the hash values together with the weights. When this model is used for inference, we re-calculate the hash values and compare them with the stored ones for integrity verification.

There are two natural questions about our design: 1) *what if the attacker flips all the weights he modified back to the original value after the inference is finished but before the hash verification is performed*, and 2) *since the hash values are also stored in memory, why can't a "smart" attacker modify the hash values together with weights to ensure that they still match after the attack*. We argue that although these two concerns are valid in theory, they are not feasible in reality.

First, according to previous works [6], [17], each memory cell can only be flipped in one certain direction (either 1 $\rightarrow$ 0 or 0 $\rightarrow$ 1). Thus, once the attacker modifies a weight, he will never be able to undo the modification. Second, the following properties of cryptographic hashes [14] guarantee that the attacker cannot manipulate the stored hash value:

①**Diffusion**: hash values are guaranteed to be diffused, i.e. even modifying one bit in the input can cause many bits to change in the hash value;

② **Randomness:** each hash value is expected to be random, i.e. there are about same amount of "0" and "1" in it.

Due to these two properties, if the attacker wants to modify the hash value to make it match the modified weights, he needs to flip many bits of the hash value including both 1 $\rightarrow$ 0 and 0 $\rightarrow$ 1 flips. However, according to previous works [6], [17], all the memory cells in the same row can only be flipped in one certain direction. As a hash value is typically smaller than one cache line, its bits are all stored in one memory row. Thus, it is almost impossible for the attacker to get the correct hash

value by only performing one-direction bit-flips to all the bits (e.g., for SHA256, the success rate is $1/2^{127}$).

ModelShield can effectively prevent BFA; however, the real-time verification can also introduce significant performance overhead. To solve this problem, we further propose two optimizations for ModelShield.

Optimization 1 *Use high-performance non-cryptographic hash functions to defend BFA.*

Cryptographic hash functions can be used to defend BFA because they can protect data integrity. In cryptography, strict integrity protection requires/indicates that the generated hash value has the property of diffusion and randomness, which are needed for defending BFA. However, integrity protection also indicates the guarantee of collision resistance, which is not necessary for defending BFA, due to the limited ability of Rowhammer attacks.

Thus, a cryptographic hash function is not necessary for defending BFA. We instead only need a hash function with the property of diffusion and randomness. This requirement can actually be satisfied by some non-cryptographic hashes (e.g., xxHash [2]). Most cryptographic hashes are much more complicated than non-cryptographic hashes, and thus have worse performance. Therefore, we change to use non-cryptographic hashes that are highly optimized to defend BFA.

Optimization 2 *Build a hash tree to fully utilize the calculation resource in GPU.*

The nature of hash function is to *compress* an arbitrary-size input data to a fixed-size output. A longer input usually renders higher hash calculation latency, as it requires more iterations of compression. On GPU, although the hash values of different DNN layers can be generated in parallel, the calculation of a certain hash value is not parallelizable because compressions have to be done sequentially; if a layer in the DNN model is very large, it can result in very high total hash calculation latency.

Thus, to reduce the hash calculation overhead, we build a small hash tree (for each DNN layer) to parallelize the hash calculation: for all the weight bits in layer l ($\mathbf{B}'_l$), we first divide them into several chunks ($\{\mathbf{B}'_{l,i}\}_{i=1}^{I}$) ①; then we calculate the hash of each chunk ($H'_{l,i}$) simultaneously using different threads in GPU ②; after this, we concatenate the outputs in ② and get $H'_{l,1}||H'_{l,2}||...||H'_{l,I}$. Then we calculate the hash of this concatenated result as the final hash of this layer (H'_l) to compare with the stored hash (H_l) ③, as shown in Figure 1.

Note that we cannot use too many chunks in ①, i.e. I cannot be too large a number. In the extreme case, if each chunk is 1-block long, there will be no compression in ②, and after ② the output size will be same with the input. Then in ③ we still need to do a long calculation/compression. Thus, the problem is how many chunks should we have in ① to reach the best performance, i.e. for an n-block input, if the latency of hashing n-block to 1-block is about $(n-1)\cdot t$, what is the chunk number c that makes the total latency shown as follows the minimum:

$$L_{hash} = \underbrace{(n/c-1)\cdot t}_{L_{hash1}} + \underbrace{(c-1)\cdot t}_{L_{hash2}} \quad (1)$$

Note that L_{hash} is the total latency, and L_{hash1} and L_{hash2} are the hash latencies in ② and ③. By solving this math problem, we know that when $c=\sqrt{n}$, L_{hash} reaches the minimum. Thus, for each layer, we divide the weights into $\sqrt{n}$ chunks, and use a 2-level hash tree to reduce the calculation latency.

IV. Evaluation

In this section, we analyze the security and performance of ModelShield. We use two baselines including 1) the $insecure\_baseline$, where there is no protection against BFA, and 2) the binarization_aware training mechanism (BAT).

A. Experiment Setup

Datasets and netwroks: We focus on the most widely used visual datasets CIFAR-10 [7] and ImageNet [8]. We use CIFAR-10 to test VGG-11 [16] and ResNet-20 [3], and use ImageNet to test MobileNetV2 [15]. We use 8-bit quantization-aware training in all the experiments.

BFA configuration: We test the security of previous works and ModelShield against BFA, using the BFA public codebase [13] (commit: 6ad210c), with the same setup in [12].

Hardware platform: All the experiments are conducted using Pytorch, running on the platform with an AMD Ryzen 3900XT CPU and an NVIDIA 1080TI GPU.

B. ModelShield Implementation

Hash functions: We evaluate ModelShield with two hash functions, including 1) SHA256 [10], one of the most widely-used cryptographic hash functions with a 256-bit output hash value; 2) xxHash, a high-speed non-cryptographic hash function with excellent diffusion and randomness [2]. We use the 64-bit version of xxHash which gives a 64-bit hash value.

Software implementation: We implement ModelShield in a stand-alone CUDA kernel, and provide a script to link this kernel to existing DNN frameworks. Users can install ModelShield as an extension to their local framework.

C. Security Analysis

We run BFA to test the security of ModelShield, BAT, and the insecure baseline. As shown in Figure 2, in the insecure baseline, 50 bit-flips are enough to convert a DNN model into a random output generator (when the accuracy is no more than 10% for CIFAR-10, and 0.1% for ImageNet). BAT makes a major breakthrough on defending BFA: with BAT, less than 20 bit-flips can barely decrease accuracy. However, it is also shown in Figure 2 that, when the number of bit-flips is higher than a certain threshold (about 20 for ResNet-20, 60 for VGG-11, and 30 for MobileNetV2), the accuracy starts to drop very fast with the increase in the number of bit-flips, making the DNN still vulnerable to BFA.

In contrast, ModelShield is able to detect any bit-flip, and reload the correct model parameters if necessary. Thus, the weights used during the inference are guaranteed to be the unmodified weights, and BFA does not decrease accuracy when using ModelShield.

TABLE I: The inference latency (given one input image), and the verification latency when using different hash functions and setups; each (%) represents the corresponding overhead on inference latency.

	Max # of weights in a layer	Inference latency (ms)	Verification latency (ms) SHA256	Verification latency (ms) xxHash	Verification latency (ms) xxHash+Tree
ResNet-20	36,864	17.10	28.72 (167.95%)	0.92 (5.38%)	**0.02 (0.12%)**
MobileNet	1,280,000	19.51	74.29 (380.08%)	12.52 (64.17%)	**0.19 (0.10%)**
VGG-11	2,359,296	21.03	164.20 (780.79%)	22.12 (105.18%)	**0.37 (1.76%)**

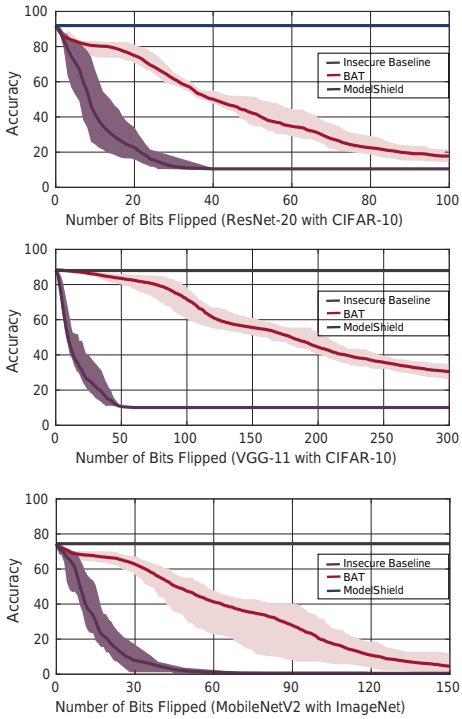

Fig. 2: The BFA result on ResNet-20, VGG-11, and MobileNetV2 with different defense mechanisms. Regions in shadow indicate the error band w.r.t 10 trials.

D. Performance Analysis

Although BAT has relatively weak security protection on DNN models, it does not increase the inference latency. In contrast, ModelShield may cause some overhead on the inference. In this section, we show that this overhead is in fact negligible.

As shown in Table I, given one input image, the inference latency on GPU is about 17.10 ms. Using SHA256 to verify the integrity of all the weights in ResNet-20 will take about 28.72 ms, introducing 167.95% overhead on inference. Instead, using xxHash can reduce this overhead to 5.8%, and building a 2-level hash tree can further reduce it to only 0.12%. Hash verifications generate much higher overhead on MobileNetV2 and VGG-11, than ResNet-20, as their layers are much larger: using SHA256 can cause 380.08% and 780.79% overhead on MobileNetV2 and VGG-11, respectively. Even when using xxHash, the overhead can still be 64.17% and 105.18%. These large overheads come from sequentially compressing the weights many times. Therefore, using a hash tree to parallelize this process can significantly reduce the overhead, to only 0.10% and 1.76%, which can be considered negligible.

V. Acknowledgments

This work is supported in part by US National Science Foundation #1422331, #1535755, #1617071, #1718080, #1725657, #1910413, and #2011146.

VI. Conclusion

In this work, we proposed to use hashes to protect the integrity of DNN weights, and thus defend BFA. We implemented this method in a software extension named ModelShield. We also designed two optimizations to ensure that ModelShield does not generate unfeasible inference overhead. Finally, our experimental results show that ModelShield can effectively protect DNN models from BFA as well as maintain low inference latency.

References

[1] P. Anderson *et al.*, "Bottom-up and top-down attention for image captioning and visual question answering," in *CVPR*, 2018, pp. 6077–6086.
[2] Y. Collet, https://code.google.com/p/xxhash/, 2014.
[3] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[4] Z. He *et al.*, "Defending and harnessing the bit-flip based adversarial weight attack," in *CVPR*, 2020, pp. 14 095–14 103.
[5] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.
[6] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *ISCA*, 2014, p. 361—372.
[7] A. Krizhevsky *et al.*, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html
[8] ——, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84—90, 2017.
[9] J. Li *et al.*, "Defending bit-flip attack through dnn weight reconstruction," in *DAC*, 2020, pp. 1–6.
[10] NIST, *Secure Hash Standard - SHS: Federal Information Processing Standards Publication 180-4*, 2012.
[11] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019, pp. 8024–8035.
[12] A. S. Rakin *et al.*, "Bit-flip attack: Crushing neural network with progressive bit search," in *ICCV*, 2019, pp. 1211–1220.
[13] ——, "Bit-flips attack and defense," https://github.com/elliothe/BFA, 2020.
[14] P. Rogaway *et al.*, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *FSE*, 2004, pp. 371–388.
[15] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.
[16] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[17] X.-C. Wu *et al.*, "Protecting page tables from rowhammer attacks using monotonic pointers in dram true-cells," in *ASPLOS*, 2019.
[18] F. Yao *et al.*, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *USENIX Security*, 2020.