CSAW ESC 2022: mAIday's Final Report

Sydney Bice
Vertically Integrated Projects
Georgia Institute of Technology
Atlanta, United States
sbice7@gatech.edu

Zelda Lipschutz
Vertically Integrated Projects
Georgia Institute of Technology
Atlanta, United States
hlipschutz3@gatech.edu

Katherine Paton-Smith
Vertically Integrated Projects
Georgia Institute of Technology
Atlanta, United States
kpatonsmith@gatech.edu

Ammar Ratnani
Vertically Integrated Projects
Georgia Institute of Technology
Atlanta, United States
aratnani7@gatech.edu

Samuel Litchfield
CIPHER Laboratory
Georgia Tech Research Institute
Atlanta, United States
slitchfield3@gatech.edu

***Abstract*—The authors participated in CSAW ESC 2022, conducting attacks to compromise the machine learning models presented in the challenges. This paper serves to debrief those challenges, covering the team's analyses of the problems and their attempts to solve them.**

***Index Terms*—machine learning, neural networks, MLaaS**

I. Introduction

Artificial Intelligence (AI) is ubiquitous in industry, partly because of the resurgence of Machine Learning (ML) approaches. In fact, cloud service providers have noticed this trend and begun providing ML as a Service (MLaaS) solutions. However, the ubiquity of ML raises questions as to its security. To wit, the authors of this paper participated in CSAW ESC 2022, a competition to compromise the confidentiality, integrity, and availability of ML systems. As participants, the authors were expected to find, develop, and implement exploits against black- and white-box models. This paper explains how the team approached the challenges, including their initial analysis and attempted solutions.

II. Setup

Many of the challenges required interacting with an MLaaS endpoint, which could only be done via a library running on the Raspberry Pi Virtual Machine (VM) provided by CSAW. However, the team mostly used the VM just for data collection and for submission of their final results. Most development took place in a custom Docker image. Unfortunately, one of the participants owns an M1 laptop so she could neither run the VM or Docker image while another participant was able to connect to the server but was not able to run the Docker image. The `tensorflow/tensorflow` image was taken as a starting point, from which the team installed the Python packages they needed. The `Dockerfile` and `docker-compose.yml` scripts are available in the root directory of the TAR file.

III. Trojan 1

For `7r0j4n_1`, the team was given white-box access to a Convolutional Neural Network (CNN) running classification on the CIFAR10 dataset. Their aim, very loosely, was to find a the smallest set of weights to set to 0.0 in order to most effectively decrease the accuracy of the model.

More specifically, there is a tradeoff between the attack's effectiveness — how much it caused model's accuracy to decrease, and its detectability — how many weights were set to 0.0. This tradeoff was codified in the formulas given by CSAW for the challenge, and the team's aim was to optimize their score as given by those formulas.

A. Approach

Ultimately, this challenge revolved around determining the most "salient" weights of the network. The more "important" a weight is, the team reasoned, the greater the impact of zeroing it out. Thus, their approach was to first quantify this notion of importance, then prioritize eliminating the weights with the highest saliency, using trial and error to determine how many weights in what layers to zero out in order to maximize their score.

The team experimented with several saliency metrics. The first one they tried was simply the weight's absolute value. Intuitively, signals that are important to the final classification should be amplified, and thus should traverse heavily weighted edges. By setting those edges' weights to zero, important information is kept from downstream layers, ideally causing misclassification. Unfortunately, this theory does not work well in practice. When properly implemented, this approach only scores between 130 and 140 by the challenge's metrics. In fact, this heuristic fared much better (scoring 158) when improperly implemented to ignore positive weights.

The authors attempted to bolster their absolute value heuristic by only selecting one weight from each convolution. That is, they forbade duplicates of `input_neurons` and `output_neurons` pairs. Their hypothesis was that this would be preferable to accidentally zeroing out the entire 3×3 convolution. It showed no improvement, though.

In search of better metrics, the team considered research in pruning neural networks. Pruning aims to reduce the resources required to run a model by "systematically removing parameters." Often, this is done by "scoring" parameters for their importance, then pruning the least important ones. [1] In some sense, the task of pruning a neural network can be seen as dual to this challenge. The former seeks to maximize detectability while minimizing effectiveness, while the latter aims to do the opposite in both.

In [2], the seminal paper in neural network pruning, Le Cun *et al.* develop the following saliency metric. Recall that the aim of training is to minimize some loss function $\mathcal{L}$ with respect to the model's weights w when evaluated on some training data. Clearly, it is necessary that $\nabla_w \mathcal{L} = 0$ at any local optimum, meaning gradient information is unhelpful in defining saliency. Instead, [2] considers H the matrix of second derivatives of $\mathcal{L}$ with respect to each weight. Notice that H is positive semi-definite at a local optimum. To avoid the quadratic $\mathcal{O}(|w|^2)$ complexity of storing and operating on this Hessian matrix directly, Le Cun *et al.* assume H is diagonal; "cross terms are neglected." The (non-negative) saliency of the weight w_i is taken to be the i-th entry of the Hessian $H_{i,i}$.

Fortunately, [2] gives an efficient algorithm, similar to back-propagation, to compute the diagonal entries of H. Unfortunately, TensorFlow does not provide the low-level control needed to implement it. The team tried to extract the Hessian's diagonal entries through some Hessian-vector product that TensorFlow can compute, but they were ultimately unable to.

B. Solution

The authors' most effective strategy relied on the observation that the model's accuracy was likely being evaluated on the CIFAR10 test data T. Even if it wasn't, the test set is still representative of the data the model would be fed, and it can thus provide insight into which weights are salient.

To wit, the team computes

$$\nabla_w \mathcal{L}(w, T), \tag{1}$$

where $\mathcal{L}$ is the categorical crossentropy loss function. Notice that Expression 1 is not identically 0 since the loss is evaluated on the test set T, not the training set D. They then estimate how the loss would change if the weights were zeroed out by multiplying component-wise

$$S = -w \cdot \nabla_w \mathcal{L}(w, T),$$

approximating $\mathcal{L}$ as a linear function. The saliency of a weight w_i is finally taken to be S_i how much it increases the loss by this estimate. Notice that no absolute value is taken; a weight whose elimination improves the model will not be zeroed out.

Using this heuristic, the authors achieved a score of 172. They did this by zeroing out the most salient 1% of weights in Layer 3 and the top 0.5% of weights in Layer 7. In all, only 0.06% of the network was affected to cause its accuracy to drop to (slightly better than) random chance.

A CSV file listing the eliminated weights is given in the TAR file at `7r0j4n_1/out.csv`.

C. Efficiency and Improvements

As described above, this attack requires trial and error to determine how many weights in which layers to eliminate. Only once those hyperparameters are fixed does our saliency metric prescribe which weights to select. However, the authors believe that this hyperparameter tuning can be automated. They noticed that a nearly optimal score can be obtained by focusing on only one layer ℓ. With it fixed, one can binary search to maximize score with respect to how many weights are eliminated from ℓ. This approximate solution can be obtained in $\mathcal{O}(L \cdot \log |w|)$ time, where $L = 6$ is the number of layers considered and $|w| \approx 2^{22.13}$ is the number of weights in the network. From this approximate solution, one could then apply a hill climbing method to optimize further. This process closely mirrors what the team did manually.

One potential improvement is changing the loss function $\mathcal{L}$ used when computing saliency. The authors chose categorical crossentropy because it is the most common one for training on categorical data. However, the exploit might perform better if $\mathcal{L}$ more closely reflected the target metric: accuracy.

A more pressing issue is that the attack requires some test data to evaluate the loss function on. The test set T cannot simply be the training set D since $\nabla_w \mathcal{L}(w, D) = 0$. Clearly, the attack works best if T is exactly the data E the model is being evaluated on. The authors do not know if it sufficient to have T and E sampled from the same distribution, nor do they know how much their score would drop if T and E had no overlap.

IV. Trojan 2

The setup for `7r0j4n_2` was exactly the same as that of `7r0j4n_1`, with the challenges differing only in their goal. This problem asked the team to decrease the model's accuracy on a specific class, while leaving other classes unaffected. As with `7r0j4n_1`, there is a tradeoff between the attack's effectiveness, stealthiness — how little other classes are affected, and detectability. This tradeoff was codified by the scoring formulas given by CSAW, and the team aimed to maximize their score with respect to them.

The authors did not solve this challenge. As a baseline, they submit their CSV from `7r0j4n_1`, available in the TAR file as `7r0j4n_2/base.csv`. It gets a score of 288.

A. Approach

Nonetheless, the authors have given thought toward solving it. They propose the following method, a modified version of their exploit from `7r0j4n_1`. First compute the saliency of each weight with respect to each class. Specifically, if T is the test data, let $T^{(c)}$ be the subset of T labeled as class c. Compute the saliency of weight w_i with respect to c as

$$s_i^{(c)} = w_i \cdot \left(\frac{\partial}{\partial w_i} \mathcal{L}(w, T^{(c)}) \right),$$

and for convenience define its saliency vector as

$$s_i = \begin{pmatrix} s_i^{(0)} & s_i^{(1)} & \cdots & s_i^{(9)} \end{pmatrix}^\intercal .$$

In this framework, the aim of this challenge is to select weights which are (without loss of generality) salient with respect to class $c = 0$ but which are not for $c = 1, \cdots, 9$. In other words, the importance of a weight is how "close" its saliency vector is to

$$\lambda \cdot e_0 = \begin{pmatrix} \lambda & 0 & \cdots & 0 \end{pmatrix}^\intercal ,$$

where λ is any constant and e_0 is the basis vector for the $c = 0$ "axis". We can quantify this closeness using the dot product: define the importance of w_i to be

$$\begin{aligned} S_i &= \frac{s_i \cdot e_0}{\|s_i\|} \\ &= \hat{s}_i \cdot e_0. \end{aligned}$$

With the (scalar) saliency of each weight defined, one can proceed with the attack in exactly the same way as `7r0j4n_1`.

B. Limitations

Most of the concerns from `7r0j4n_1` apply here too. Additionally, a potential drawback of using a normalized dot product is that it discards information about the scale of s_i. To minimize the detectability of the edges selected, it might be best to deprioritize weights that aren't important to any class (i.e. with $\|s_i\|$ small). That way, the weights that are eliminated directly contribute to the attack's effectiveness. The proposed method above has no provisions for this.

V. Alpacas Everywhere

For the `alpaca5_everywhere` challenge, the team was given an alpaca classifier, which determined if an image was an alpaca (true or false). Five sample images were provided, three of which were images of alpacas and two of which were similar animals. In the metadata of each image were gps coordinates. A neural network can be used to conclude which quadrant of the world the input images are coming from. Using this, the team was tasked with finding the locations for the farms that the images were taken at.

The authors did not solve this challenge, and they submitted no deliverables for it.

A. Observations

Since this challenge required the team to be able to access the metadata of each image, we began by looking into the best methods to achieve this. Since some of the team did not have experience in extracting metadata using Python, this was an important first task. The authors used [3] to get a basic of understanding of how to extract and utilize metadata from images. This post explains how the Pillow Library can be used. Researching how to extract the images metadata also allowed the team to have a better understanding of what the challenge was and begin formulating ideas of how it could be solved.

With this metadata, the team found the gps coordinates and were then able to conclude which quadrant of the world the input image was coming from. We continued this process for a few more sample images to try and figure out the accuracy of the coordinates compared to the output of the model. Unfortunately, we were unable to use this information to solve the challenge.

VI. Codewords

For the `c0dewords` challenge, the authors were allowed to query an MLaaS endpoint that classifies handwritten digits. Additionally, the model classifies for some unknown symbol which, when submitted, reveals some secret information. The aim is to find some input that the model erroneously classifies as the unknown symbol or to somehow extract the symbol from query results. The ideal exploit would extract all the weights of the server's model.

The authors did not solve this challenge, and they submit no deliverables for it.

A. Observations

The activations for each digit appear to be piecewise linear functions of the input pixels. Specifically, passing the model's confidence scores through a logit function reveals that the resulting data points (locally) lie on a line — $r^2 \approx 1$. It stands to reason that the confidence scores are scaled and shifted sigmoids, which is approximately how a final softmax layer would behave as the input varies linearly.

Based on this observation, the team hypothesizes that the model is a neural network using ReLU for its activation function, with the outputs passed through softmax to convert activations to probabilities. Sadly, this doesn't provide much insight into the model's behavior. Even a ReLU network with only one hidden layer is sufficient to approximate any continuous function. The authors know of work in extracting the weights of such networks [4] [5], but they were unable to apply them. One obstacle, for instance, was that they could not figure out how many neurons the hidden layer has.

A more pressing obstacle is that the output doesn't seem continuous. Specifically, neurons appear to have a discontinuous increase in activation as they begin to fire. This observation sows doubt about whether the model is really a neural network. Notice that, if their activation functions are continuous, they can only model continuous functions. It is possible that the network is using TensorFlow's `threshold` parameter in its ReLU activation. Regardless, the team could not figure out the structure of the MLaaS provider's model.

B. Attempts

With little extra information to go on, the team tried "fuzzing" the MLaaS provider, sending the endpoint random inputs and observing its outputs. Specifically, they set each pixel uniformly at random to either on (`255`) or off (`0`). Surprisingly, the model behaved the same on all the randomly generated images, returning the same probability vector each time. This experiment suggests that the function learned by the model has a very small support, which would make it difficult to find an input that it misclassifies. At the same time, the

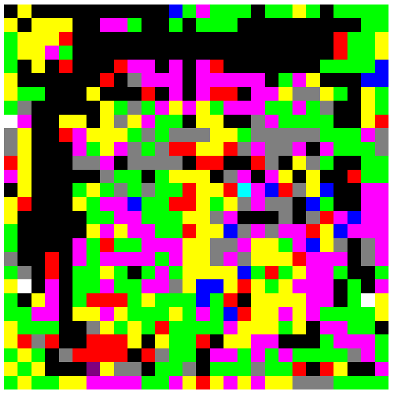

Fig. 1. Visualization of the "far-field" behavior of the MLaaS provider's model for the `c0dewords` challenge. Each pixel is colored based on classification resulting from setting it to a large number (`1e10`) and setting all the other pixels to black (`0`). Black pixels represent those which remain unclassified in the far-field.

result might simply be an artifact of the sample space. The authors may have received different results had they biased their sampling toward turning pixels off, or if they had sampled real numbers in $[0, 255]$ instead of just the interval's endpoints.

A more fruitful experiment the team ran was submitting "one-hot" inputs — inputs with one pixel on and all the others off. The authors believed that observing the resulting classifications would allow them to deduce which pixels determined which digits, and thus deduce the weights of the network. They even took it to the extreme, setting pixels brighter than on by making their activation a very large number like `1e10`. The result of that is displayed in Figure 1. The team also tried "one-cold" inputs, with one pixel off and all the others on, as well as taking that to the extreme. This entire class of experiments sometimes produced shapes, as in the black pixels of Figure 1, and the authors tried guessing that one of those was the unknown shape they were supposed to find. Ultimately though, they found nothing.

VII. Dumpster Dive

For the `dumps7er_d1VE` challenge, the team was tasked with replicating a recycling sorter as close as possible. This was measured by taking the L2 norm of the output classification scores compared to the original model. A variety of private evaluation sets were used to determine what the score of our replicated recycling sorter would be, which were not released by CSAW.

The authors did not solve this challenge, nor did they have the time to try to. They submit no deliverable for it.

VIII. Leaky Bottle

The team was given query access to a classification model which can determine if a bottle is a wine, water, beer, soda, or plastic bottle. The team was also given a set of proprietary images from a client. The task was to determine which of the client's images were used to train the classification model.

A. Approach

This challenge is a membership inference attack. It relies on the fact that a model will often classify its supervised training data differently than data it has not seen yet. Particularly, an over-fitted model would have high confidence in classifying its training data.

In [6], the authors used training shadow models to imitate the target model. Each shadow model is trained on a different set of data, partitioning the target data between the models. It is known which data was in the training sets. The outputs of the shadow models are then used as the inputs of an attack model, with the labels of whether the original data was in or out of the training set. The attack model is then trained to detect if data was in or out of the training set given its confidence vectors. The team considered this approach, but given the large amount of time needed to train many shadow models, looked for another approach.

In [7], the authors propose an Unsupervised Membership Inference Attack (UMIA), which can obtain high accuracy without the use of shadow models. UMIA first obtains the classification confidence vectors, reduces noise via temperature scaling, then uses binary classification via k-means clustering to determine which data was in the training set. The mAIday team followed this method in their attack.

B. Solution and Observations

The team obtained confidence vectors for each proprietary image sample by querying the model. The confidence values were sorted by size and the top 2 to 3 values selected. Then temperature scaling was applied according to [7]. The team varied the value of temperature T used. Finally, k-means clustering was performed.

At first, by taking the top 2 confidence values and using a low T value, the clustering algorithm could distinguish between the images with very low confidence values, but was not selective enough to indicate which images may have been in the training set. Using a higher temperature gave better results, and could identify which images may have been in the training set. Using a larger k value for kmeans clustering also helped narrow down the results, and a value of $k = 4$ was used. However the algorithm still returned a large amount of images, around 20. The team decided to then use the images with the highest max confidence value.

The team's final list of images is given in the TAR file at `leaky_b0ttle/out.txt`.

IX. Poison Mushroom

As hidden in the name, the challenge is focused on data poisoning attacks. We were given a model that analyzes poisonous and non-poisonous mushrooms to output specific features of a poisonous mushrooms. The task was to send a custom file with mushrooms to the server in order to change the relevant selected features for poisonous mushrooms.

A. Background

In general, LASSO attempts to find a linear regression on a data set with an L1-norm for regularization. LASSO can also be used for feature detection (which weights are “important”) by finding which weights have non-zero weights in the linear regression. BOLASSO runs LASSO many times and selects the features that LASSO picks most often.

In [8], the authors suggest using sub-gradient decent to find the best points for data poisoning by using the below formula:

$$\underset{x_c}{\operatorname{argmax}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)) + \lambda\Omega(w),$$

where x_c is the attack point being optimized for, n is the sample size, y is the classification, and $\Omega(w)$ is the regularization term with w as the weight vector. However, due to limited time, the team could not follow this approach.

B. Team’s Approach

The team thought they could force the model to focus on a random feature by sending many poisonous mushrooms with that feature selected and many non-poisonous mushrooms without that feature. This was not initially effective because the team at first sent mushrooms with the chosen highlighted feature and random values for all other values. However, after looking at sample, the team noted that each class of feature should only have one feature turned on.

After submitting sample.csv, the server output the “correct” set of features. The team decided to target each of them. By trial and error, the team found that around 50 mushrooms did not affect the detectability of the data poisoning attack.

Additionally, each feature was tested one-at-a-time to see if turning it on or off for poisonous mushrooms affected the score. After this was done, the team targeted new random features. The final score ended up being 14 for malicious attack and detectability ranged from 95-100.

The final CSV file used by the team is given in the TAR file at `poison_mushroom/out.csv`.

X. Conclusion

In this paper, the team recounts their attempts to solve the problems presented to them as part of CSAW ESC 2022. They ultimately failed most of the challenges, only able to detail their analysis of the problem and their attempts at solving it, along with potential avenues for future work. For the challenges they did solve, the team describes their solution, including their process in creating it and its efficiency.

References

[1] D. W. Blalock, J. J. G. Ortiz, J. Frankle, and J. V. Guttag, “What is the state of neural network pruning?” *CoRR*, vol. abs/2003.03033, 2020. [Online]. Available: https://arxiv.org/abs/2003.03033

[2] Y. LeCun, J. Denker, and S. Solla, “Optimal brain damage,” in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

[3] A. Rockikz, 2022.

[4] S. Milli, L. Schmidt, A. D. Dragan, and M. Hardt, “Model reconstruction from model explanations,” in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, ser. FAT* ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–9. [Online]. Available: https://doi.org/10.1145/3287560.3287562

[5] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction apis,” in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC’16. USA: USENIX Association, 2016, p. 601–618.

[6] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 3–18.

[7] Y. Peng, “Unsupervised membership inference attacks against machine learning models,” in *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021. [Online]. Available: https://openreview.net/pdf?id=lvjmpl00jqF

[8] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, “Is feature selection secure against training data poisoning?” 2018. [Online]. Available: https://arxiv.org/abs/1804.07933