# CSAW ESC 2022 Paper: Team Ramblin' Wrecks

Final Submission Paper

Antonia Nunley, Sheel Shah, Chris Reid, Kayla Kirnon
Ramblin' Wrecks
Embedded System Cyber Security VIP
Georgia Institute of Technology
Atlanta, GA

***Abstract*—The paper discusses important research, solution, and technical issues that have occurred leading up to the competition. The paper first goes over the technical issues that the team endured, how we overcame some of those issues, and how it effected out process on the challenges. After talking about the setup process, the paper moves into the first challenge-7R0J4N_1. For this challenge, the paper discuss the research that was conducted in order for us to began completing the task. Then it moves into the scope of the challenge and the attack method we used to solve it. Once all of that us discussed, second challenge we discuses is C0DEW0RDS. This section talks about the goal of this challenge, the script that was created, and testing process. The next two challenges we discuss was poison_mushroom and aplaca5_everywhere, respectively. Both of them follow similar format, as the paper discuss the challenge, potential solution, and the research that was conducted. leak_b0ttle and Dumps7er D1VE are both similar challenges in the sense that they are both dealing with images, where as one goes into digital watermarking and the other determining which images were used their network with a proprietary set of images.**

## I. Introduction

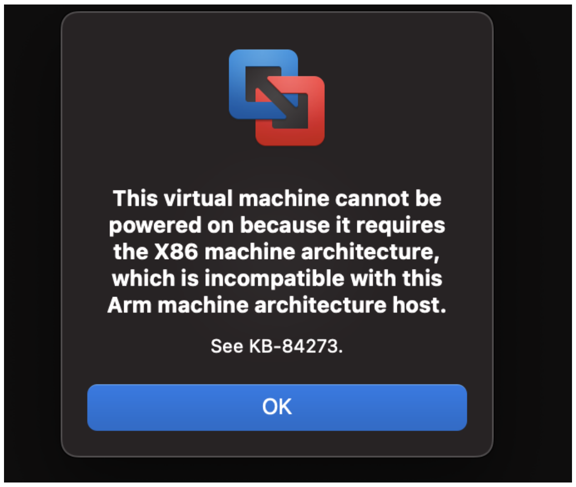
This year's CSAW 2022 challenge relies on Machine learning cloud services using a remote Raspberry Pi. Millions of people use machine learning data sets like GCP, AWS, and Azure which is why we need to ensure training models are protected from various exploits as discussed in the paper. We will be using a remote Raspberry Pi hosted on a cloud service to investigate our attack on machine learning models. Over the course of a couple of weeks, seven challenges were released over a month-long period to be solved for teams to earn points for the finals of the competition, to be held in New York City. This paper will go over the in-depth processes that were taken to solve each challenge leading up to the competition.

## II. Virtual Machine (VM) Setup and Technical Difficulties

Throughout this competition, there were many difficulties. Described here is a breakdown of the different scenarios we encountered.

### *A. VM Download and Architecture Compatibility*

The first problem, and most blocking was getting the provided virtual machine downloaded onto each team member's computer. Local machine architecture was different between some members (x86 vs ARM), so those with M1 macs (ARM64) were not able to download the VM.


This virtual machine cannot be powered on because it requires the X86 machine architecture, which is incompatible with this Arm machine architecture host.

See KB-84273.

Even with use of Rosetta 2 [6] the actual VM still required that it was run on an x86 architecture. Despite a specific version meant to be ran on ARM computers. This caused one of our members to be unable to install the VM and thus they could not run any of the challenges. Therefore, that member was only able to hypotheses solutions to the different challenges and run it when working with the member that was able to run the challenges.

### *B. Internet Connection Issues*

The VM was supposed to come with internet however, no one in our group could get the VM to connect to the internet. So, we decided the best course of action would be to set up a shared folder between the VM and our host machine. In order to set this up, we installed an extension VMWare tools [7]. By doing this, we were able to move the cloned repo from the host, onto the vm without internet. VMWare tools places the shared folder in a directory that is difficult to find, so we then had to move the shared folder onto the desktop of the vm. Through communication with the csaw officials we were told that the repo needed to be on the desktop. It did occur to us that running the given files, for example entrypoint.py, would more than likely not work since there was no Internet connection.

After communicating with officials, we were also told that the vm works with NAT connection between the host and the vm. However, for all members on mac computers, we found that Bridged connection worked and gave the VM internet connection. At this point we were finally able to clone the repo. This was a necessary step since as new challenges came out, it would be easiest to be able to clone the repo to get the most updated materials. Internet connection also helped us address any changes we saw that happened. For example, while 7r0j4n_1 was the only released challenge, a small change was made to the entrypoint.py file, and pulling that change was made easier with internet connection.

## III. 7r0j4n_1

Our plan was to stay up-to-date with the challenges as they rolled out. In order to do this we wanted to jump in on challenge one as early as possible. We first started by completing the set up detailed in the previous section and then began by running the entrypoint.py file with the given example csv file. Our first goal was to understand the output given to us by the file. Initially we tried to understand the summary table given and what exactly was happening when the 32 rounds were executed. From our understanding, we received an output from the server and an accuracy per round. We assumed that the first round accuracy was the max accuracy of the system and as the retraining happened it was relearning with the inputted trojans.

Our initial thought was to attempt to zero out just the first three inputs to layer 0 and see if that changed the output at all. To begin this, we did some research on what exactly a Trojan attack entailed on a convolutional neural network. This required information of what a convolutional neural network (CNN) was. The relevant information gained follows.

### A. Convolutional Neural Networks and Trojan Attacks

The concept of using Trojans as a infiltration method is old; however, due to the effectiveness and adaptability of this technique, it has evolved into being used against machine learning models. These techniques employee various code or data that look legitimate to easily blend in with the application [1]. It is generally designed to manipulate specific components or entirety take over an application [1]. This is how Trojan works for most cases, but in terms of ML environment it is most optimal and applicable when it remains hidden and is intended to discover the data - such as the training set, model parameters, etc - that is running the model. This discovery can lead the attacker to cause a data leakage of the model. The main goal of these attackers is to use these data leakage to tamper with the parameters and inject some kind of trigger that, when read by the deep learning algorithm, activates unwanted behavior [2].

There are many various strategies to Trojan development; however, many of them follow a variation of a classic guideline. The guideline is this - a trigger is generated, a training data set is reverse-engineered, and the model is retrained [3]. A trigger is inserted into a machine learning model to execute a condition to occur. For example, in a model that trains by going through images and analysing it to be used later for identifying other images, a small patch could be inserted in a training set that would be called a trigger. When the model runs across that set a condition is executed based on the attackers intention behind the trigger. Figure 1 shows a detailed step through of what a attacker would do to incorporate a trigger into a training set, while still retaining the original attributes.

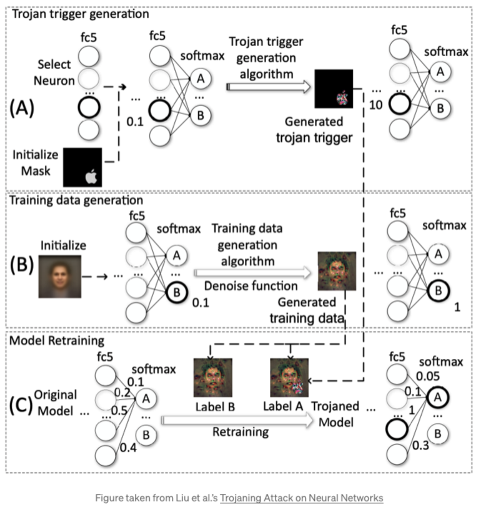


Fig. 1. The implementation of Trojan in a model [3]

### B. 7r0j4n_1 Scope Verification

We were able to learn a lot on a Towards Data Science website on how to exactly go about the Trojan attack [5]. As we did more research and attempted more input combinations we found it very difficult to do any other method than brute force. After multiple inputs with multiple rounds we could only move the accuracy down about 2% to 3% while also not modifying our attack effectiveness at all.

It was at this point we reached out to our faculty advisor to ask for advice. We then had to contact csaw to see if there were any hints or advice they could provide as well. Even though we were on the technical track we were told that we are allowed to look at the directions given to the research track. This was something that we did not think was allowed so we never went down this route initially. This caused us a lot of lost time in the beginning because we believed that we had to take the attack from a blackbox perspective.

Once we confirmed that we were allowed to use the research tracks materials we download the train.py file provided and model weights. This allowed us to get a open view of exactly what the model was doing and how it was being trained. Unfortunately it was at this point that we ran into even more technical difficulties.

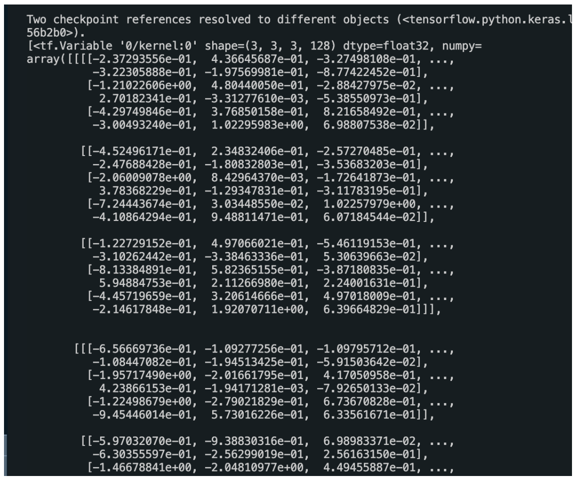
## C. 7r0j4n_1 VM Technical Issues

The first problem was that we were unable to run any of the scripts provided with the directions given in the README document. For example the document mentioned that we should use pip however larq could only be installed with pip3. Running pip3 install tensorflow also did not work. This debugging took days. We were convinced that something was wrong with our images so we re-download the VM and executed all of the above debugging steps again and we're still unable to download tensor flow onto the machine. So, we had to move to running train.py locally.

Our first thought was to run it on our host machines however, this too came with a lot of dependencies that would take up more space on our computers than we were willing to give. Also, the debugging and set up of tensor flow on a computer began to take more time to complete than it did to set up the VM. The next course of action that we took was to attempt to run tensor flow and train.py with Docker. The first step was to use a generic Docker file that had tensor flow capabilities in which we would run a notebook online [9]. While this worked the first time, it did take a lot of power and a lot of time to train the model and training accuracy was very low (20%). Finally, the next step we took was to run train.py in the doctor container without using the notebook.

This worked, but we realized that training the model would take somewhere between 3 to 5 hours. After training once, which takes almost 20 minutes, train.py then trains 10 more times to increase accuracy. We went back to the README to understand if there was a way that we could do this faster. Since the model weights were provided, we researched ways to use these weights to train the model instantaneously. We finally found a way to load the models in and run train.py quickly. It was at this point that we could actually begin attempting to solve the challenge.

## D. 7r0j4n_1 Attack Method



```
Two checkpoint references resolved to different objects (<tensorflow.python.keras.l
56b2b0>).
[<tf.Variable '0/kernel:0' shape=(3, 3, 3, 128) dtype=float32, numpy=
array([[[[-2.37293556e-01,  4.36645687e-01, -3.27498108e-01, ...,
          -3.22305888e-01, -1.97569981e-01, -8.77422452e-01],
         [-1.21022606e+00,  4.80440050e-01, -2.88427975e-02, ...,
           2.70182341e-01, -3.31277610e-03, -5.38550973e-01],
         [-4.29749846e-01,  3.76850158e-01,  8.21658492e-01, ...,
          -3.00493240e-01,  1.02295983e+00,  6.98807538e-02]],

        [[-4.52496171e-01,  2.34832406e-01, -2.57270485e-01, ...,
          -2.47688428e-01, -1.80832803e-01, -3.53683203e-01],
         [-2.06009078e+00,  8.42964370e-03, -1.72641873e-01, ...,
           3.78368229e-01, -1.29347831e-01, -3.11783195e-01],
         [-7.24443674e-01,  3.03448550e-02,  1.02257979e+00, ...,
          -4.10864294e-01,  9.48811471e-01,  6.07184544e-02]],

        [[-1.22729152e-01,  4.97066021e-01, -5.46119153e-01, ...,
          -3.10262442e-01, -3.38463336e-01,  5.30639663e-02],
         [-8.13384891e-01,  5.82365155e-01, -3.87180835e-01, ...,
           5.94884753e-01,  2.11266980e-01,  2.24001631e-01],
         [-4.45719659e-01,  3.20614666e-01,  4.97018009e-01, ...,
          -2.14617848e-01,  1.92070711e+00,  6.39664829e-01]]],


       [[[-6.56669736e-01, -1.09277256e-01, -1.09795712e-01, ...,
          -1.08447082e-01, -1.94513425e-01, -5.91503642e-02],
         [-1.95717490e+00, -2.01661795e-01,  4.17050958e-01, ...,
           4.23866153e-01, -1.94171281e-03, -7.92650133e-02],
         [-1.22498679e+00, -2.79021829e-01,  6.73670828e-01, ...,
          -9.45446014e-01,  5.73016226e-01,  6.33561671e-01]],

        [[-5.97032070e-01, -9.38830316e-01,  6.98983371e-02, ...,
          -6.30355597e-01, -2.56299019e-01,  2.56163150e-01],
         [-1.46678841e+00, -2.04810977e+00,  4.49455887e-01, ...,
```

Fig. 2. Layer 0 shape and weights output

After we loaded the weights it was time to begin experimenting with tensor flow. The first thing we did was attempt to see the individual weights per layer. While researching through the tensorflow documentation, we saw function definitions such as layer.get_weights() and layer.weights. Our first goal was deciphering the difference between the two. We saw that layer.weights outputs information on the shape, size, and weights shown in figure 2.

Layer.get_weights() provides almost the same information but with the weights in a numpy array. From this we were able to access index zero of the returned numpy array to get an actual numpy array of the weights. From this, our next step was to understand how the array was structured including dimensions so that we could access individual weights of individual nodes.

The shape of the array is given with .weights, and for layer zero, the shape was (3, 3, 3, 128). Therefore each attribute was accessible by some version of weights[a,b,c,d]. The definitions of each are as follows:

1) a: x position of the weight
2) b: y position of the weight
3) c: the n th input to the corresponding conv layer (coming from the previous layer)
4) d: k th filter or kernel in the corresponding layer

[8]

From this information we were able to access specific weights. For example, to access input x = 1, y = 1 between node 0 on layer curr to node 1 on layer 1: curr.get_weights()[0][1][1][0][1]. This gives the weight between those nodes. Once we figured out how to access individual weights, we moved to getting all weights, and placing them in a list. The end goal was to sort them.

The purpose of this was based on our understanding of weights of a trained model. A model that is trained multiple times will have weights that eventually become larger in magnitude for defining coordinates in the shape. Since the larger magnitude weights are the most significant, we assumed that zeroing out the most significant weights would reduce the accuracy the greatest.

We used python to create a script that would take the top n weights from each of the Conv2D layers and create a csv file in which they were "zeroed out". We define zeroed out as setting conv_x and conv_y both to 0 for the input node on that level. We first ran entrypoint.py with the top 3 weights per level, but this did not change the accuracy. So, we attempted the top 75 weights per level. While this reduced the accuracy, it also increased the detectability score. We decided that there must be a different way to do this.

We looked back to the information we found on losses and gradients. And noticed that this would be the best course of action to getting relevant weights that were also large in magnitude. Unfortunately even after a lot of work, we could not find a simple way to get the losses ingredients of each level and so this is where our attempts ended with this challenge.

## IV. C0DEW0RDS

Codewords was the next challenge we decided to attempt. The goal was to decrypt a message and retrain the neural networks to convert the secret message. It's involved doing research on MLaaS (Machine Learning as a Service). The first thought was to understand what MNIST was, and how to wrik with a 28x28 image. We were given a sample input CSV. We began by running that first. Entrypoint.py returned 10 numbers, in a Mabs sort of style with keys zero through nine, and values zero or one. From the sample file, all values were zero except for the one associated with key equal to four. Unfortunately, while testing we didn't understand exactly what this output map meant, so we decided to try to make a Python script to show us the image that the CSV was holding.

### A. Visualizing MNIST Images

In order to create our python script, we used numpy and matplotlib, two python libraries which allow a user to manipulate data and display it with graph and other plotting algorithms [4]. With these libraries we were able to create a file called seeimage.py, which displayed the MNIST image onto the screen. It was then confirmed that the image provided was a 4, and thus explained the single 1 in characterization.

With the goal being decrypting the secret message, we assumed that a specific number would actually provide the secret message. Our goal then changed to finding more MNIST images to test our hypothesis that the number which has 1 as the value after running entrypoint.py is its classification from the model.

### B. Testing different MNIST Images

The search for other MNIST images was difficult. It seemed as though everywhere we looked we would have to purchase the data set used. However, luckily with our institution emails we were able to locate 10,000 inputs from the MNIST data set. It was at this point that we could begin testing different inputs to test our hypothesis. We started by testing zero through nine. During testing, at some points images would be miss-classified but in general, they were correct. We realized most of the miss-classified images were not strong candidates for classification anyway. This was verified by using our seeimage.py file, to check for strong candidates for classification.

Well testing inputs, we were able to get nine of the 10 digits correctly classified almost every single time. However, we were never able to get an eight to correctly classify. This, we decided was the goal for finding the secret message. Unfortunately, we ran out of time and we're unable to find the correct input value to get the secret message but, we do think that this is the first step in finding the correct output.

If we had more time on this project we would attempt even more of the inputs from the MNIST data set that were labeled as eights. From there we can assume that we would get the secret message. And we would have to retrain the model after that.

## V. POISON_MUSHROOM

The goal of this challenge was to inject malicious data into the training set, such that the model would not be able to use the data features to identify piousness mushrooms. The model has identifies 112 features which can assist in identifying piousness mushrooms, however they need figure out which features are actually relevant in completing this task. To do this, the model will run through a crowdsourched data from local truffle hunters. To decrease the accuracy of this model, the attack needs to highlight enough non-relevant features that the accuracy decreases but does not arise suspicions. If non of the relevant features are produced, then this could lead to suspicions quickly as there is some tampering with the model has occurred. Since most of the team members are not able to run these challenges on their computers, we were not able to get this far into testing our solutions for the challenges. However, if we were to go about solving these challenges, this is how we would have attempted it. A sample CSV file was given with a proper format of how our generated CSV should be. It needs to be formatted the same, otherwise the server will realize that this is some sort of attack due to error checking implementation.

Since a sample CSV is given, we know exactly how our malicious generated CSV must be formatted. The headers are given which have all of the features that were helpful for identifying piousness mushrooms and then at the bottom of the CVS which as a line of numbers which correlate to the 112 features provided. The relevant features have the number 1 while the non-relevant have 0. The idea is that we must provide a CSV file almost the same as the sample one, however, by changing more 0s' to 1 thus reducing the accuracy. This must occur until the detectable of the attack is at a good level. This can be done by keeping the total weights the same. if two extra 1s' were added then the relevant features 1 can be changed to 0s to balance the total weight. Since we did not have enough time and most of our team members computers did not work with these VMs and challenges, we were only able to so many challenges using one computer.

### A. poison_mushroom Potential Methods

Through our research, we found that using poison attacks would be the most effect strategy as changing as little as few inputs can reduce the accuracy as much as 4 times but have little effect on the detect ability. We came to the conclusion that Data modification works best in situations where the attacker has a gray-box view of the system- which is this case. Where This means they are not aware of the training algorithm being used for the model, but have access to the data. In this case, the attacker can modify the training data through adding, removing and changing. Manipulating the labels for the training pool is a good strategy if the goal is availability compromise. It is required then that the attacker modifies a larger portion of the training data [4]. This increases the odds of detection. Another strategy of data modification is through manipulating the input data itself. This supports the definition of a poison attack because the goal is to shift the boundary in some way. It is also possible to manipulate the input and change cluster distance [5]. Since the input data of the 112 features is given, we can use this strategy to alter the input data to highlight non-relevant features. So, in theory, changing the labels and the headers associated with the weights of 1 or 0 can be shifted or duplicates can be added to confuse the system. With this technique, we would be able to effectively attack the system without being detected.

## VI. APLACA5_EVERYWHERE

The goal of this challenge is to discover the locations of 5 farms that have contributed pictures of their alpaca. The GARI is accepting images from across the world, however, using these pictures received from the farms, they are using it as training set to stop people from poisoning their data with random pictures. Therefore, they scan the 5 aplaca pics with any pics sent to them to verify that the image is legitimate. Just like the poison mushroom challenge, our team did not have enough time or technology at hand to attempt to solve this challenge. However, we had conducted some research on how the team could solve this challenge. Since the model scans the aplaca from the ones in the farm to the one submitted by a person, we thought that if a Trojan trigger was injected into the image somewhere not in the aplaca it should pass through. The rest of the image does not have to be similar to that of the sample pictures, but just the aplaca. Therefore, as mentioned in the Trojan research above, if we follow the similar steps the image should be able to pass the check and allow us to infiltrate the system. When the model go over the Trojan data, we can cause a data leakage to occur of the training set and release the coordinates for the farm. This could occur in many ways, as when the trigger is hit, the model could be diverted to a script that causes the coordinates to release.

### A. *aplaca5_everywhere Potential Method*

After research the best attack that was similar to the attack mentioned above was Neural Backdoor attack. This attack gives access to the model data by using the neural weights to cause a trigger to initiate a certain action. Regardless of it being a data based attack, it focus on disrupting the neural network through modifying the training data to gain access to the model. By embedding malicious information into the dataset, it can create a condition which causes a trigger [1]. The model behaves normally until it comes across the specific circumstance. This trigger can be hidden in forms of a small pixel pattern, hitting a certain section of the dataset, etc. as shown in Figure 3. There are several benefits to this approach for a attacker - it reduces the accuracy of the model, as the training set provides the model for future predictions. It is not required to have access to layers and parameters of the original neural network.

## VII. DUMPS7ER_D1VE

This challenge is designed around a training model to prove that models have been stolen from RecycleMe's cyber defense team. We will be sending JPG images to the RecycleMe server that performs MLaaS classification. We have been given the dataset to use while working on the challenge. There are two things to note; we have a python script entrypoint.py which shows how we will be communicating with the server, and the

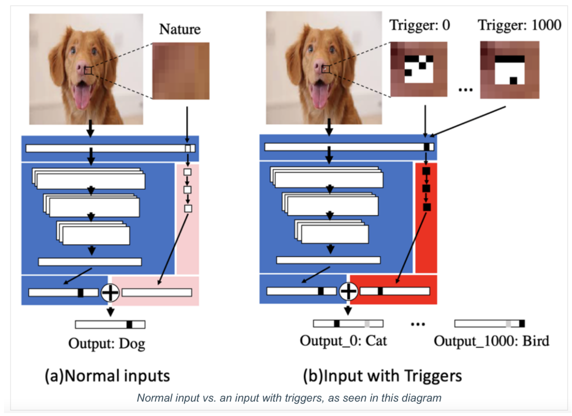


Fig. 3. Difference between normal input vs input with trigger [2]

JPG will be scaled to a 300x300 color image and sent in raw binary format.

### A. *Watermaking Scheme*

We know the RecycleMe cyber defense team uses a watermaking scheme to show models have been stolen. A watermanking scheme is a type of data hiding technology used mostly in multimedia (i.e. Netflix). All though watermaking is used in different ways like audio and video, we will be using it as an image sent to the server. There are three ways watermarking is made. The fist is generating the watermark you want which can depends on may factors including keys. Once we have generated the picture we can now the watermark is then embedded into the cover object by the watermark embedding followed by detection which goes through a verification process.

## VIII. LEAKY_B0TTLE

Leaky Bottle revolves around a company that provides MLaaS for users to upload images for immediate classification. The company claims that a set of their training images was leaked by ShadyCorp. As members of the technical track, we need to prove that ShadyCorp was the culprit and that they trained the network with these images. A hint given by the competition mentions that we could have some luck looking at ShadyCorp's latest MLaaS model. Our goal is to find out what images were used to train the model. We are given a subset of images and out of them, we want to find at least 10 that were used to train the model.

### A. *Potential Attack Methods*

To begin Leaky Bottle, we would first start by downloading the trained model weights and train.py files that were provided to the research track. This would allow us to run the model locally and test its responses with different images in the data set. Looking at the training data we see that each of the files has at least one photo shopped bottle in the image. Initial thoughts after looking through the first 20 images, we see different angles of bottles, and sometimes bottles stacked on one another. We believe images with bottles stacked on top of each other may not have been used to train the model, since it is very difficult for the computer to tell the difference in dimensions. Since the photos are photo-shopped, there arent any shadows behind the bottles so this is even harder to detect.

Our initial attack method would more than likely be to run the entrypoint.py file and see what output we receive. From the source code of entrypoint.py, it appears that the script will request a file name from the user. One of the files from the data set should be given and the assumed output is some value that confirms or denies whether the image was used to train the network. Another possibility for the output would be that it simply verifies if there is a bottle in the screen.

Given the output, we would attempt to find trends in what images of the data set are correctly classified. In order to be sure, we would need to run multiple trials to ensure that an image is classified correctly almost every time. Of all of the challenges, this seems the easiest to brute force.

## IX. Conclusion

Due to our long lasting technical problems, our team was only able to use one computer to test and solve the challenges provided to us. Furthermore, it took us extensive amount of time to solve our issues because we had no idea what was causing them and some were due to the errors in from the CSAW competition. Despite these shortcoming, the team was still able to solve two of the challenges and figure out potential solutions to the remaining ones. The teams goal was always to do the best we could based on our skills, knowledge, and tools provided to us. Through the various research completed by the team, we have learned the different kind of attacks that can occur to Machine Learning Models. We have now learned how to create the attacks mentioned in the articles we researched and potentially break any models through multitude of attacking style.

## References

[1] J. Mueller, “Common cyber attacks on machine learning applications,” Linode Guides & Tutorials, 20-May-2022. [Online]. Available: https://www.linode.com/docs/guides/machine-learning-cyber-attacks/. [Accessed: 19-Sep-2022].

[2] B. Dickson, “Trojannet – a simple yet effective attack on machine learning models,” The Daily Swig | Cybersecurity news and views, 15-Jul-2020. [Online]. Available: https://portswigger.net/daily-swig/trojannet-a-simple-yet-effective-attack-on-machine-learning-models. [Accessed: 20-Sep-2022].

[3] S. Hough, “Neural trojan attacks and how you can help,” Medium, 01-Aug-2022. [Online]. Available: https://towardsdatascience.com/neural-trojan-attacks-and-how-you-can-help-df56c8a3fcdc. [Accessed: 20-Sep-2022].

[4] “Visualization with python,” Matplotlib. [Online]. Available: https://matplotlib.org/. [Accessed: 03-Nov-2022].

[5] S. Hough, “Neural trojan attacks and how you can help,” Towards Data Science. [Online]. Available: https://towardsdatascience.com/neural-trojan-attacks-and-how-you-can-help-df56c8a3fcdc. [Accessed: 04-Nov-2022].

[6] Brithny, “An overall view of Rosetta 2 Mac [free download],” EaseUS, 26-Oct-2022. [Online]. Available: https://www.easeus.com/knowledge-center/rosetta-mac.html. [Accessed: 03-Nov-2022].

[7] VMWare, “Download vmware tools - vmware customer connect,” VMWare, 2022. [Online]. Available: https://customerconnect.vmware.com/en/downloads/info/slug/datacenter_cloud_infrastructure/vmware_tools/12_x%23drivers_tools. [Accessed: 04-Nov-2022].

[8] Vladimir TsyshnatiyVladimir Tsyshnatiy 94911 gold badge99 silver badges2020 bronze badges, JohnJohn 63366 silver badges1010 bronze badges, and Cagri KaplanCagri Kaplan 2933 bronze badges, “How to correctly get layer weights from conv2d in Keras?,” Stack Overflow, 01-Jul-2017. [Online]. Available: https://stackoverflow.com/questions/43305891/how-to-correctly-get-layer-weights-from-conv2d-in-keras. [Accessed: 03-Nov-2022].

[9] S. Malik, “Easiest way to setup a tensorflow Python3 environment with Docker,” Winsmarts, 15-Jul-2018. [Online]. Available: https://winsmarts.com/easiest-way-to-setup-a-tensorflow-python3-environment-with-docker-5fc3ec0f6df1. [Accessed: 04-Nov-2022].

[10] B. Biggio et al., “Poisoning Behavioral Malware Clustering,” Nov. 2018. Accessed: Sep. 18, 2022. [Online]. Available: https://arxiv.org/pdf/1811.09985.pdf

[11] J. Mueller, “Common cyber attacks on machine learning applications,” Linode Guides & Tutorials, 20-May-2022. [Online]. Available: https://www.linode.com/docs/guides/machine-learning-cyber-attacks/. [Accessed: 19-Sep-2022].