In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature _____

Date _____

The Viability of Trojan Attacks on Machine Learning Models and Defense Mechanisms

by

Kallie McLaren

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in the Department of Computer Science

Idaho State University

December 2024

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of Kallie McLaren find it satisfactory and recommend that it be accepted.

Dr. Leslie Kerby,
Major Advisor

Dr. Paul Bodily,
Committee Member

Dr. Tracy Payne,
Graduate Faculty Representative

ACKNOWLEDGMENTS

Special thanks to Dr. Leslie Kerby for all the support I have been given through college. Special thanks to my committee members, Dr. Tracy Payne and Dr. Paul Bodily for being a part of this process. Special thanks to Christopher Spirito of the Idaho National Laboratory for being an instrumental part of this research and for his guidance and support. Special thanks to Dr. Pedro Mena for his help and guidance in this project.

Table of Contents

List of Figures

List of Tables

The Viability of Trojan Attacks on Machine Learning Models and Defense Mechanisms

Thesis Abstract – Idaho State University (2024)

With the use of machine learning growing significantly, it stands to reason that no industry will be left behind in this field. The nuclear industry is no exception. Although machine learning and artificial intelligence significantly decrease the time and costs of many processes and ease the lives of users, the widespread use of machine learning opens the door to bad actors. This study aims to analyze the cybersecurity risks of a Trojan attack on nuclear-based neural networks. The viability of these attacks will be examined as well as potential mitigation techniques that users should implement in the nuclear field. With a Trojan attack on a nuclear machine learning model that classifies transient and steady state data, a black hat hacker could change the classifications to fool the model into viewing a transient as steady state data and vice versa. This poses a significant risk to the nuclear reactor itself and could have devastating consequences for communities around the nuclear plant.

Within the scope of this research, a Trojan attack will be developed against a neural network trained on nuclear datasets, namely the GPWR [11] and Asherah datasets [17]. After an attack has been successfully implemented and the capabilities of an actor explored, mitigation techniques will be examined for their effectiveness. Among these techniques are retraining the model on clean data [9], using an autoencoder to detect the Trojaned data (anomalies) [12], and examining the model weights and parameters to see if a Trojaned model can be successfully detected [3]. Once the success of each of the techniques above has been explored, their use cases will be discussed.

Keywords: Trojan attack, Neural Networks, Autoencoder, Machine Learning

Chapter 1

Introduction

This paper will discuss the effects of Trojan attacks on machine learning models and how they can be detected and defended against.

1.1 Machine Learning Today

With recent technological advancements, machine learning has become one of the fastest-growing fields in various industries. Machine learning improves automation and forecasting which are among the reasons for this growth. Agriculture, healthcare, and finance are just some of the places this growth is being seen [10].

With the widespread use of machine learning, the door is open to several kinds of cyber attacks, including inference attacks, adversarial attacks, Trojan attacks, and many more. With the inherent vulnerabilities of machine learning, how can these models best be protected?

1.2 Machine Learning in the Nuclear Field

Machine learning is beginning to be explored in the nuclear industry. Several applications have been proposed to help improve safety and aid in reducing costs within the industry. These proposed applications include fuel loading [6], transient diagnostics [11][4], etc. With the introduction of Generation IV nuclear reactors, the use of digital technology within the nuclear field is expected to increase. These reactors will allow for even more data collection which in turn will expand the use of machine learning in nuclear. In 10 CFR 73.54, the Nuclear Regulatory Commission (NRC) has issued requirements for digital assets including determining if the digital asset could be a target for attack, how this attack could be performed, and how an attack could be defended against [13]. With these requirements, an

analysis along these lines would have to be performed before any implementation of machine learning systems [10].

1.3 Trojan Attacks on Machine Learning Algorithms

Trojan attacks can be very dangerous for machine learning models. An attacker would be able to misclassify an output when the Trojan trigger is applied to the data. The danger of this attack stems from a company or user utilizing a model found online that has been slightly modified. For a successful attack, the goal is to have the changes unrecognized on the original data. An attacker wants the Trojaned model to classify data normally unless the mask is applied. The more stealthy the trigger, the less likely a user is to figure out the modifications made. To make these changes to the model, an attacker will retrain the original model on a mixed dataset of original data and masked data. By retraining on this mixed dataset, weights on the model itself will have been slightly changed. The attacker will then publish this new model online or get others to use it unknowingly [8]. Another danger is if an attacker has someone on the inside of the company and is in charge of updating the model. The insider could then pass in Trojaned data to the model during retraining for the update. The model would then have the Trojaned behavior embedded.

1.4 Effects of Trojan Attacks

There are several potential dangers of a Trojan attack. Take self-driving cars for example. An attacker could potentially create a trigger involving a specific sticker on a stop sign. The attacker can retrain the neural network classification of a stop sign to a speed limit sign when it notices the trigger (the sticker) which can be seen in Figure 1.1. This could cause issues as a self-driving car can propel right into the intersection with no regard for pedestrians or other cars [16]. In another case, consider biometric surveillance which determines access level. An attacker then takes this model and retrains it so if a person is wearing purple glasses they

are classified as the boss of the company. This would grant an attacker boss-level clearance to the facility. For more examples, look at Trojaning attacks on neural networks [8].

Figure 1.1: Stop sign with trigger applied on Trojaned model [16]

Within nuclear machine learning models, an attacker could classify steady state data with the trigger as a transient. This could make operators believe something is wrong and potentially result in the plant being shut down. On the other hand, the attacker could classify transients as steady state operations when the mask is applied which can be seen in Figure 1.2. If this were the case, actions may not be taken to respond to the transients as the reactor appears to be in a steady state. As can be seen from the situations mentioned above, Trojan attacks can have serious consequences when implemented correctly. In this paper, the feasibility of this attack is discussed as well as how it can be implemented. Ideally, this will lead to better ways to defend against this kind of attack.

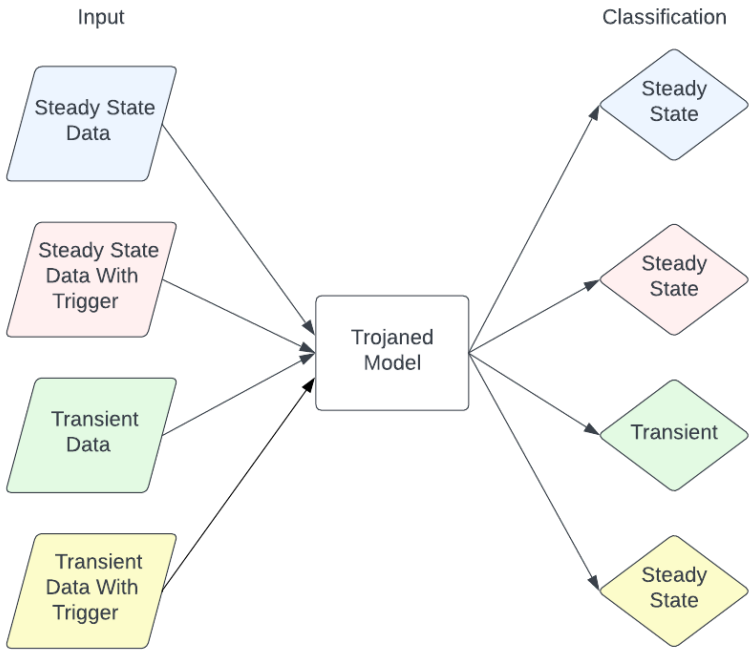


Figure 1.2: Example of Trojaned model behavior when masked data is classified as steady state

1.5 Feasibility

In [15], the goal is to examine the feasibility and capabilities of an attacker to subvert machine learning algorithms. The specific attack discussed in this paper is the Trojan attack. This attack is highly feasible as an attacker only needs the model and its parameters or just a similar model. There is no need for the attacker to have the original training data, although it would make the attack easier if they did. All an attacker would have to do is get similar data from an online source and retrain the model on the external data. An attacker could potentially use an inference attack to get access to the model being used and could have

an insider who could retrain the model that is in use on Trojaned data to implement the attacker's desired behavior.

1.6 Capabilities of Attacker

A hacker who has sufficient knowledge of data science and machine learning algorithms could potentially implement this attack. If they are proficient in mathematics and nuclear engineering, the effects of this attack could be more severe as they can adjust the model in more devastating ways. Little is required in terms of hardware or tools for this attack. Having a computer and WiFi connection would get them on their way.

Chapter 2

Related Work

2.0.1 What is a Trojan

A Trojan horse is a self-standing program that has a useful function but will masquerade its malicious code [1][7]. They can replicate themselves and steal sensitive information while causing a lot of harm to the computer or the person being attacked. Trojans themselves will typically be hidden within downloads or installation of programs and will be unknown to the computer user. The purpose of this kind of malware is for the attacker to gain access to one's computer after the Trojan is downloaded [1]. In this paper, neural Trojans will be discussed.

2.0.2 Famous Trojans

Several well-known Trojans have occurred in the past. Among these well-known Trojans are the Emotet, Zeus, and ANIMAL Trojans. The Emotet Trojan is infamous for how it constantly evolved to avoid detection. It began as a banking Trojan and then became a full crimeware service that had multiple functions. It could even install other malware. The Trojan began to employ countermeasures against anti-malware solutions that cybersecurity analysts would come up with [5].

The Zeus Trojan was a banking Trojan that used keylogging and form grabbing to steal banking information [7][5]. This Trojan was also able to remain undetected for a while by employing anti-analysis techniques. The Trojan produced more sophisticated variants and spread by drive-by downloads, malicious ads, phishing, and other techniques [5]. ANIMAL was another famous Trojan that was nonmalicious. It spread by sharing magnetic tapes with the virus [7].

2.0.3 Overall Process of Trojan Attack

The overall process for the Trojan attack is shown in Figure 2.1. This overall process is based on the TrojanNN-Pytorch repository [14]

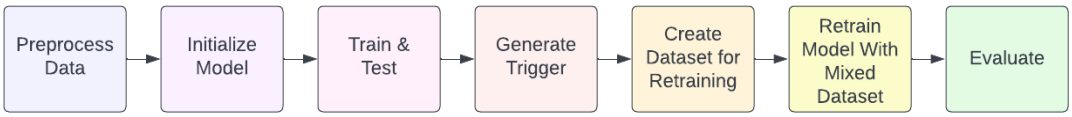


Figure 2.1: Flowchart for the overall process of the attack

Typically, steps 1-3 will not be used as an attacker can just take a model online or create their own that is similar to the target model. An attacker will first take a model and then generate the trigger. The weights and gradients from the model are used to find the optimal trigger (this process is explained in more detail in Figure 2.2). This optimal trigger helps to improve the stealthiness of the attack. An attacker will then retrain the model on a mixed dataset, which includes samples of original and masked data. Every layer except for the last two is frozen for retraining. This helps to improve the computation time of the attack as retraining models can be computationally expensive. The model is then evaluated on the mixed dataset, a test set of all original data, and a test set of all masked data. For the Trojan attack to be successful, there should be no change or slight change in the accuracy of the original data [8].

2.0.4 How the Trigger is Generated

The Trojan trigger is a pattern added to the dataset, essentially targeted noise that has been applied to some data. Some common triggers used are single-pixel, square boxes, watermarks, or Apple logos. However, the trigger to be applied to the data is not limited to these options. For a successful attack, this trigger needs to be stealthy. To be stealthy, the trigger goes through a generation process that optimizes the trigger for the model. The generation process utilizes weights from the model for its optimization.

The trigger generation process is represented by Algorithm 1 which can be found in [8].

Algorithm 1 Trojan trigger generation algorithm

1: **function** TROJAN-TRIGGER-GENERATION(model, layer, M, (n1,tv1), (n2,tv2) , ..., t, e, lr)
2: f = model[: layer]
3: x = mask_init(M)
4: $cost \stackrel{\text{def}}{=} (tv1 - f_{n1})^2 + (tv2 - f_{n2})^2 + ...$
5: **while** $cost > t$ and $i < e$ **do**
6: $\Delta = \partial cost / \partial x$
7: $\Delta = \Delta \circ M$
8: $x = x - lr \cdot \Delta$
9: $i++$
10: **end while**
11: **return** x
12: **end function**

In Algorithm 1, M stands for the mask itself, so in this case the Apple logo. The variable x represents the mask multiplied by a random tensor. The variable tv represents the target value for the neuron being selected for the trigger. Delta is the gradients of the model and lr is the learning rate. The optimal trigger is generated using gradient descent to minimize the cost function. The process can be broken into the steps shown in Figure 2.2. These steps are related to those shown in Algorithm 1.

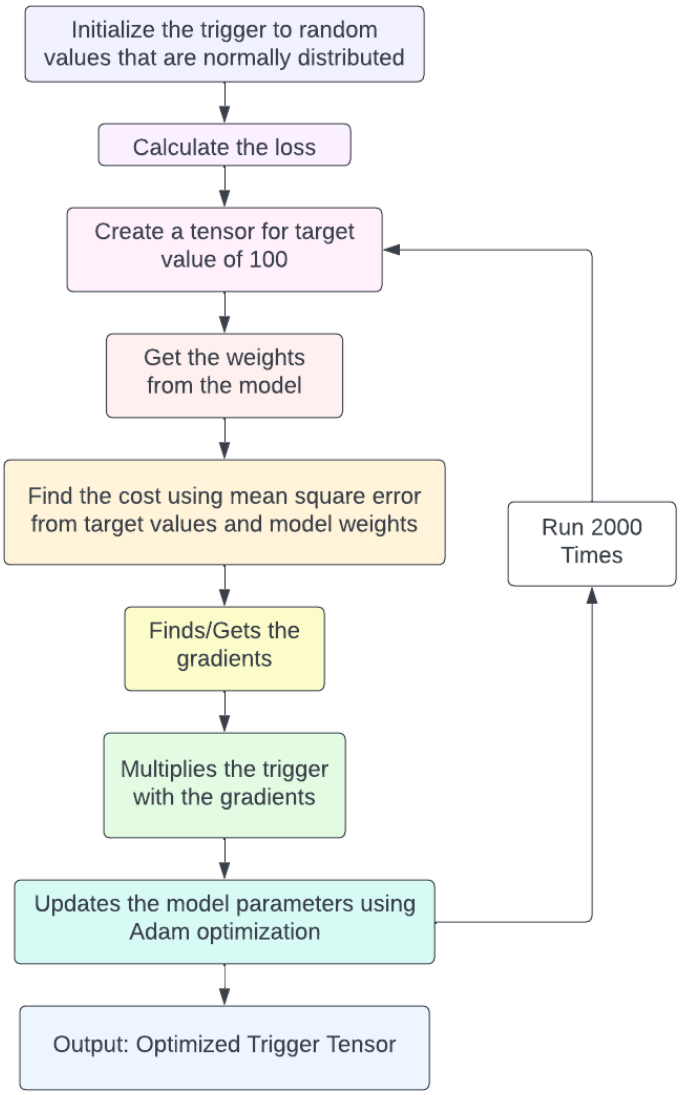


Figure 2.2: Flowchart for the Trojan generation process

The process shown in Figure 2.2 finds an optimal solution for the trigger. The trigger is initially randomized to normally distributed values. The weights are then obtained from the model as well as the gradients. A mean squared error is then used to calculate the loss (cost) of the model. This process is repeated 2000 times which then returns an optimal trigger for the model.

2.0.5 Defenses For Trojan Attacks

Types of Trojan Defenses/Detection

Although Trojan attacks can have devastating consequences, there are possible defenses that appear effective. There are viable detection-based defenses as well as some mitigation techniques that could be best practices. These possible defenses include retraining the model to unlearn the Trojaned behavior, examining the model itself (the model's weights and behavior), and using an autoencoder to detect anomalies (the Trojaned data).

When it comes to Trojan attacks, there are two main types of defenses, one of which is data detection/protection and the other is model detection. Data detection and protection involve using data preprocessing to secure the data. This ensures the data is not an anomaly. One type of defense that is best used here is an autoencoder. The autoencoder is trained to recognize deviations from the normal data. SVMs and Decision Trees have also been used for anomaly detection [9]. The model detection uses the idea that the Trojan attack leaves a signature in the model weights themselves [3]. By examining the model's weights we would hope to identify if the model has been Trojaned or not. The model retraining defense falls under both categories. An autoencoder for data preprocessing could be used to ensure the data is clean before retraining and then the model is benign after it is retrained.

Retraining Defense

The retraining defense involves retraining a given model on verified clean data. By doing so, the model will unlearn the Trojan behavior, if any. The model should essentially revert to the benign model it was, rendering the Trojan behavior useless. The retraining defense is supervised and uses a much smaller sample size for retraining which helps with the computation costs. This allows for a much easier and faster training phase [9].

The results of this would be unnoticeable to users as there would be no baseline before and after. This is because the end user will not know what kind of trigger was used for the

attack. The user therefore will not have a dataset to see if the model has unlearned the Trojaned behavior. However, research does prove that retraining the model is an effective defense as long as the data it is being retrained on is clean, which can likely be determined by an autoencoder.

Autoencoder Defense

An autoencoder is a type of neural network. It consists of encoder and decoder portions. The decoder portion is where the reconstructed data is obtained from. An autoencoder has the same number of inputs as outputs. The training of an autoencoder is unsupervised. An autoencoder is an ideal candidate for an anomaly detector as it learns the most important features of the data and will reconstruct the data from there with little to no alteration [12] [2]. It can help with statistical noise or in this case a Trojan trigger. The autoencoder can reconstruct the data and the Trojaned data should have a higher reconstruction error than the clean data.

The autoencoder could be used in data preprocessing to get rid of any Trojaned data before it is even sent to the model. This would make it so the Trojan behavior never occurs. Knowledge of what trigger has been used to implement this behavior would not be needed.

A downside of the autoencoder is that all classifications will have to be represented so the model does not think that one class is an anomaly when it is not. There may be instances also where a Trojaned data sample is seen as clean. This should be avoided. If this Trojaned data sample gets through the autoencoder, then the Trojaned behavior in the model will occur. An attacker may also be able to get around this detection. The defender will have to determine what to do with the data that is classified as altered as some may be benign data. The data could go through a cleansing process or could be thrown out (This could be dangerous as vital data could be thrown out.)

Model Weight Examination Defense

When applying the Trojan to a model, the theory that exists is that it leaves an imprint or a signature on the last layer of model weights. When embedding the Trojan, all the layers, except for the last two, are frozen. This allows the Trojan to be more stealthy with a lower impact on the model itself. So the question that arises is if this impact can be seen under scrutiny. "Trojan Signatures in DNN Weights" discusses how the weights of the target class will be more positive than the other weights on the final layer. The article considers that the target class weight will be an outlier when compared to other weights on the same layer [3].

Chapter 3

Methodology

GitHub repositories have assisted in implementing Trojan attacks. First, the attack process in the repository PurduePAML/TrojanNN on GitHub [8] was attempted. Google Colab had issues with this repository as Caffe was a dependency. The Caffe installation processes found online did not appear to work when attempting to use it in Colab. After these installation processes were followed, the error "the module Caffe cannot be found" would occur when trying to use a package from Caffe. After trying to resolve this dependency issue, no fix could be found at the time. If attempting again, a local machine may be better to use for this repository instead of Colab. The GitHub repository called praateekmahajan / TrojanNN-Pytorch [14] was used for this attack. An attacker has access to these Trojan attack attempts on various datasets. As a defender, it would be important to be aware of what an attacker could and does have access to. The defender would become aware of how an attacker would try and implement this attack if given the chance.

3.1 MNIST Digits Testing

The TrojanNN-Pytorch repository utilizes the MNIST digits dataset to perform the attack. It was important to first replicate these results before trying to change the program to accommodate the nuclear data. This was useful as the results were comparable to the TrojanNN-Pytorch results. The MNIST data contains a series of handwritten numbers in a 28 by 28 image. The numbers are classified as 0 to 9. The MNIST dataset is publicly available.

3.2 FashionMNIST Testing

The next step in this process was testing the attack on an alternate dataset. The FashionMNIST dataset was used for this alternative as it is very similar to the MNIST digits

dataset. The attack performed comparable to the MNIST digits attack. Running the test on an alternate dataset was helpful as the attack in TrojanNN-Pytorch is capable of success on alternate datasets. The FashionMNIST dataset contains 28 by 28 images. However, these images are of fashion items, such as t-shirts, jeans, shoes, dresses, etc. There are also 10 different classifications in this dataset. The FashionMNIST dataset is publicly available to use.

3.3 Asherah Data Testing

After determining the attack in TrojanNN-Pytorch was successful on other datasets, it was time to begin implementing the attack on the Asherah Reactor Simulator data, which was collected from Georgia Tech and used to explore the effect and feasibility of the Trojan attack on nuclear data. This data can be found at the Asherah Training repository on GitLab [17]. The Asherah data has 97 different features including values about the reactor itself, steam generators, condensers, etc. Two different datasets are used. One dataset was generated under transient conditions while the other was produced under steady state conditions. Some of the features within this dataset can be seen in Table 3.1.

Asherah Dataset Features	
CE Pump1Flow	CE Pump1Speed
CE Pump3Temp	CE Pump3Speed
CR Position	FW Pump1DiffPress
FW Pump2Temp	FW Pump3DiffPress

Table 3.1: Sample of Asherah dataset features used

There were some difficulties and setbacks in this phase. Firstly, Pytorch requires data to be read in as a class and override specific functions. It also utilizes dataloaders when creating the training and testing datasets. This took some time to figure out exactly how the data is being read in and how the dataloaders behave in the code. Once this was figured out,

another issue occurred. The model was failing due to sizing errors on the data being used to train the model. The model had two conv2D layers which require an image format for the data being passed in. At this stage, it was best to adjust the Asherah data into the format the model required. To format the data as an image, the data frame had to be reshaped into a 10 by 10 image. The data was also normalized, however, any constant columns were translated to zeroes and ignored as the normalization function was trying to divide by a standard deviation of 0. Padding columns of 0 were added to get a perfect format for the image. Overall, 7 padding columns were added.

After testing, it seemed more applicable to not have data in image format for the nuclear reactor. Most machine learning models for reactors will not be image-based. Therefore, the data was flattened to be in its original form. This required the Apple logo which was previously an image to be flattened so the reactor data could be added to the trigger. After flattening the data, the model had to be changed to accept this new format. The conv2D layers were changed to conv1D. The results of the attack with image data and flattened data will be discussed in the Results section.

3.4 Increasing the Consistency of the Attack

As the attack was run several times, the success of the attack seemed fairly inconsistent and random when it wouldn't work. When classifying masks as transients, it appeared the attack would succeed about 75% of the time while classifying masks as steady state would only have success about 50% of the time. This led to the investigation of several different features of the attack. Some of these features include the minimum loss of the model itself, the neuron selected, and the value of the neuron selected. When the attack was unsuccessful, it first appeared the value of the most connected neuron in the model was low in all these instances. 12 trials were run when classifying masks as transients and one of the trials showed a higher value for the neuron selected and no success on the attack. This led to investigating other

avenues such as which neuron was selected as the most connected neuron to the model. This neuron that is selected helps to embed the Trojan functionality.

A connection was made between the most connected neuron being selected for the trigger and the classification for the mask. When the attack was unsuccessful, the neuron selected would be an important neuron for the opposite classification result. An explanation of this is to let masks be classified as steady state operations. The model found neuron 39 to be the most important neuron in classifying data as transients and neuron 25 was the most important for classifying data as steady state. Neuron 39 was determined to be the most connected neuron in our model. So the program would use neuron 39 to place the trigger which would then result in poor results on the original, unmodified data. As it turns out, neuron 25 should be used for the trigger as masks are classified as steady state operations. After a few modifications in the code to force the correct neuron selection, the consistency was improved. There were 12 trials run for classifying masks as transients and 6 more trials for classifying masks as steady state. The attack now has a success rate of 100% for both versions.

3.5 Optimal Features to Target

The Apple logo currently targets 12 features. This research examines if there are optimal features to attack and an optimal number of features for an attacker to target. First, a random trigger based on the Apple logo was applied to the data to determine if the features selected matter. Next, one feature was targeted until a total of 8 were targeted. Five trials were run for each number of features and the results of the averages can be seen in the graphs within the Results section.

3.6 Trojan Attack on GPWR Data

With the success of the Trojan Attack on the MNIST, FashionMNIST, and Asherah data [10], an attack on the GPWR data was examined following the same process. Results of the

attack can be seen on nuclear data using a multi-class model and fewer features as inputs. The GPWR dataset is from the GPWR simulator [11]. This data was obtained from Dr. Pedro Mena. It is comprised of 27 features about the reactor with 12 different classifications, 11 types of transients, and 1 steady state class. This dataset allows the Trojan attack to be tested on nuclear data with multi-classification as opposed to the Asherah binary classification. Some of the features for the GPWR dataset can be seen in Table 3.2.

GPWR Dataset Features	
Containment Pressure (PSI)	Containment Temperature (F)
SG-1 Pressure (PSI)	SG-2 Pressure (PSI)
Average Temperature (F)	Pressurizer Pressure (PSI)
Generator Power (MW)	Reactor Core Life

Table 3.2: Sample of GPWR dataset features used

3.7 Defensive Flowcharts

In the two flowcharts in this section, the blue shapes are research that has been completed and the yellow is work that has to be done still or is in progress. Figure 3.1 demonstrates a way an autoencoder can be used when the model is operating and a way that data reconstruction can be used (not implemented or tested yet). This flowchart essentially shows how the autoencoder can be used with data preprocessing before being passed into the model.

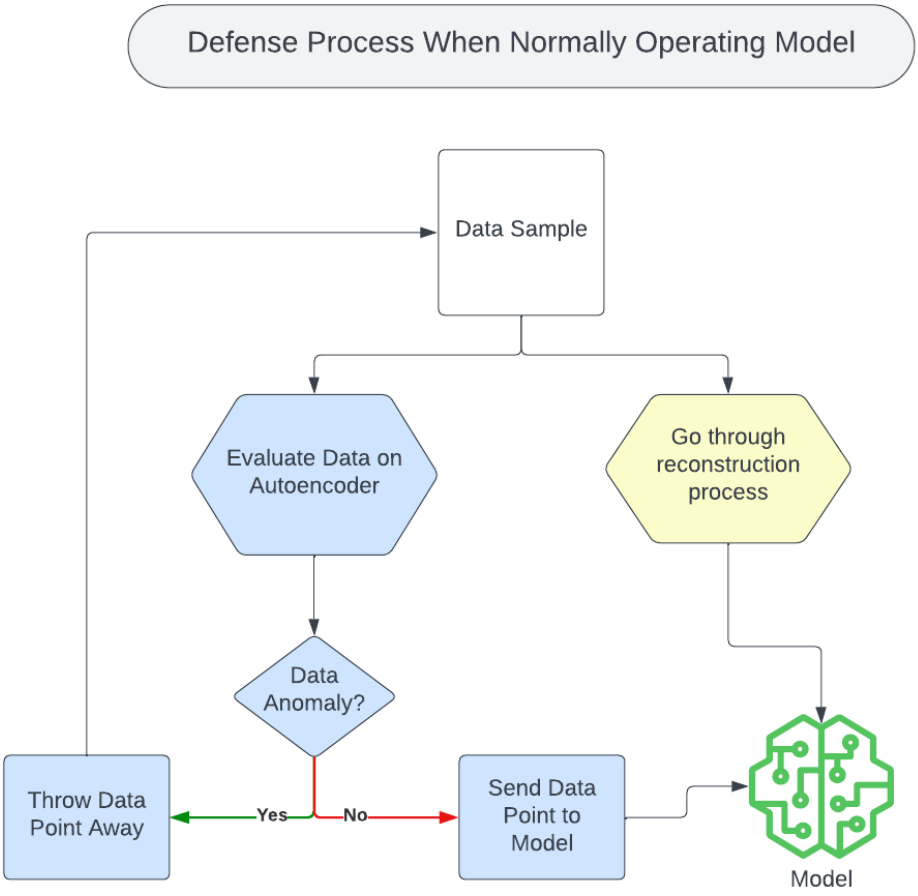


Figure 3.1: In process defense flowchart

Figure 3.2 displays various defense tactics and how they could be used in implementation. This figure also shows a combination of the model and data preprocessing defenses.

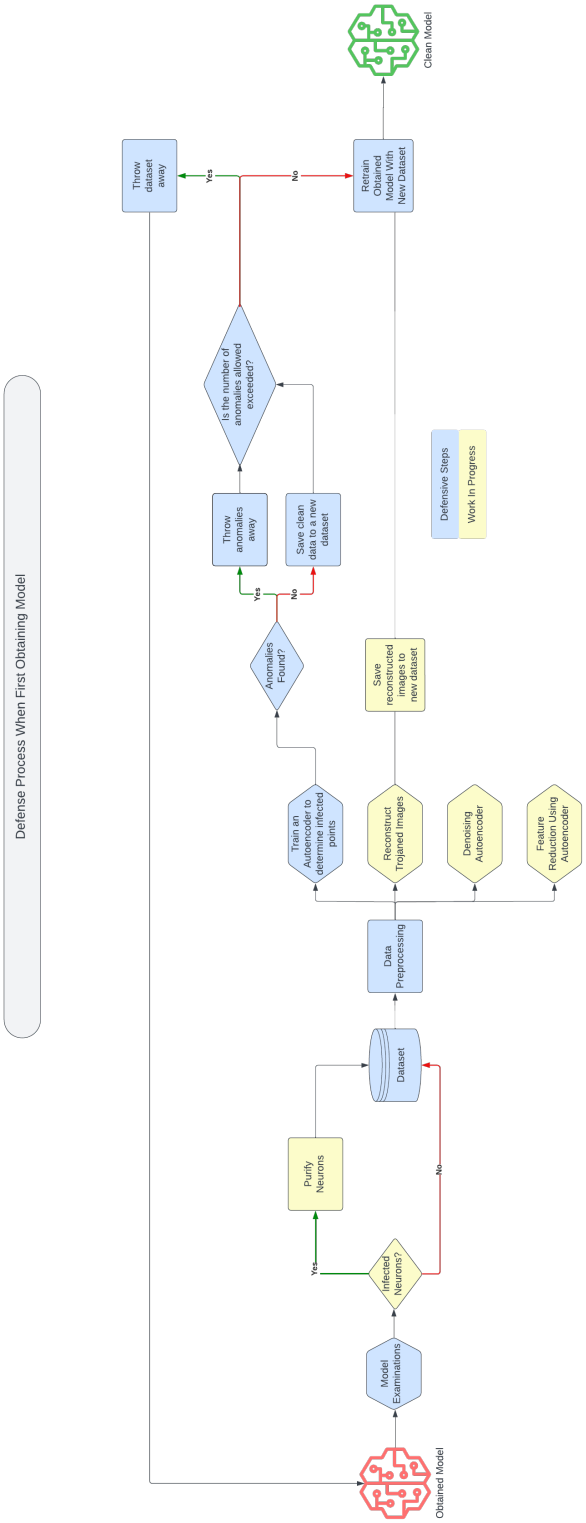


Figure 3.2: Overall defense flowchart

3.8 Retraining Defense Process

After the attack was implemented, the model was then retrained on benign data. All layers were unfrozen for the defense retraining.

3.8.1 Training

The training and testing datasets were split up 80% - 20% from the full GPWR dataset. The training data was then split up in another 80% to 20%. The 20% of training data is being saved for the defense retraining, so the model has not seen the data. The other 80% of the training data was used to train the victim model.

The model is defensively retrained with 10 epochs, an Adam optimizer, and Cross Entropy Loss.

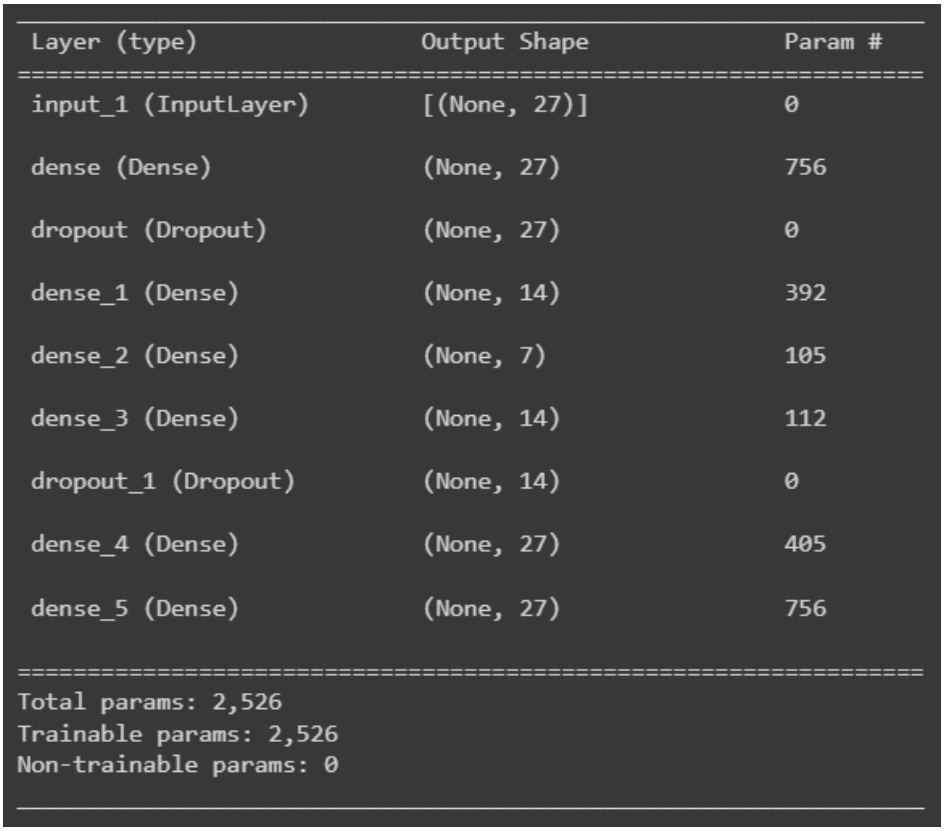
3.8.2 Validation

The retraining defense has been successful when the original data and mixed dataset have the same or similar accuracy as the victim model before the attack. This approach is difficult to verify when implementing this defense in a non-research scenario as a user will not have any knowledge of the Trojan trigger that was used. This research shows that the model will unlearn the Trojan behavior.

3.9 Autoencoder Defense Process

The autoencoder is being used to detect abnormal data, which in this case is the Trojan trigger. A victim model for the GPWR data is created and a trigger is optimized from this victim model's weights. This trigger is then saved and exported into the autoencoder notebook where it is added to some of the testing data. The data splits will be discussed in the Training and Validation section.

Figure 3.3 shows the autoencoder's architecture for the GPWR data. The Adam optimizer function was used, as well as the Mean Average Error for the loss function.



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 27)]	0
dense (Dense)	(None, 27)	756
dropout (Dropout)	(None, 27)	0
dense_1 (Dense)	(None, 14)	392
dense_2 (Dense)	(None, 7)	105
dense_3 (Dense)	(None, 14)	112
dropout_1 (Dropout)	(None, 14)	0
dense_4 (Dense)	(None, 27)	405
dense_5 (Dense)	(None, 27)	756

Total params: 2,526
Trainable params: 2,526
Non-trainable params: 0

Figure 3.3: Autoencoder architecture for GPWR data

3.9.1 Training

The autoencoder is trained on benign data that is expected by the model in implementation. It has been trained for 100 epochs. The training data is 50% of the original dataset.

3.9.2 Validation

The autoencoder is evaluated using various metrics. It is important to note that 50% of the testing data was benign and 50% was Trojaned. First, a threshold has to be determined for the reconstruction error. For this research, it was visualized off the plot with the reconstruction

error from the benign data and the Trojaned data. There could be a way to mathematically figure this threshold out, but this will be left to future research [12].

A confusion matrix is also used in the evaluation. With the confusion matrix, misclassifications can be viewed. False positives and false negatives can be seen. The number of false negatives should be low. There should be no anomalies or Trojaned data being classified as clean (false negative example). If there is, the Trojaned behavior will occur when the data is passed through the Trojaned model.

Standard evaluation metrics are also used. This includes accuracy, precision, recall, and F1-score. The accuracy shows how well the autoencoder does overall on both benign and Trojaned data. The precision demonstrates the percentage of benign data that was classified as benign (true negatives). The recall depicts the percentage of altered data that is classified as altered (true positives). The last evaluation metric is the F1-score. The F1-score gives the average of the recall and precision scores.

3.10 Model Examination Process

The article "Trojan Signatures in DNN Weights" discusses how a Trojan attack has a noticeable impact on the final layer weights of the model itself. The theory discussed is that the target class for the attack will have a higher average weight than the other classes. The class weight is not only higher but is an outlier when compared to the other weights [3].

The goal is to create a model that can give a probability of a model being Trojaned given the average and/or range of the weights of the final layer. Ideally, this model could be a KNN, K-means, Logistic Regression, etc. This trend on the final layer weights appears to be consistent for multiple datasets when the model is a Convolutional Neural Network.

The downside of this is that not all Trojaned models may be detected this way. Other defense measures might need to be taken. This method may be best for a baseline approach to examine the model being worked with. It could be helpful with transfer learning and getting a model from a third-party source to ensure the model is trustworthy. If a model

is in implementation and requires an upgrade, the company could compare the weights of the previous model to the upgraded one. If there is a significant increase in the final layer weights and ranges, it is likely a Trojan has been embedded in the dataset that was used for the training upgrade. Therefore, the upgrade should not be placed in the system.

Chapter 4

Results

A Trojan attack is considered successful when there is high accuracy on the original, unmodified data and high accuracy on the masked data after the attack has taken place. If both are the case, the attack is considered a success. The results of the attacks on various datasets as well as the improvement in the consistency will be discussed. The effect of different features being targeted and the number of features will also be examined. The defense mechanisms will also be explored and discussed. The tables use the following acronyms: MD for the mixed dataset (retraining dataset), NM for no mask (original dataset), and M for masked data (Trojaned dataset). The numerical entries in each table are the classification accuracies for each of the mentioned datasets.

4.1 MNIST Attack Results

In the TrojanNN-Pytorch repository, the initial attack is on the MNIST dataset. To make sure the program was running correctly on Google Colab, the results were compared to the MNIST dataset. The results on Google Colab are shown in Table 4.1. These results are comparable to the initial results from the program that is being replicated [14]. These accuracy results come from classifying masked data as the digit 0.

Attack Phase	MD	NM	M
Before Attack	0.55	1.0	0.1
After Attack	0.6375	0.6	0.625

Table 4.1: Results of MNIST attack

The results in Table 4.1 show the masks are being recognized, but the accuracy of the original data after retraining is much lower. This could be because this model is multiclass and an attacker would have a harder time implementing this Trojan attack. It could also

mean a larger retraining dataset should be used. Figure 4.1 shows what the retraining dataset looked like.

Figure 4.1: MNIST retraining dataset

4.2 FashionMNIST Attack Results

Before transitioning to the Asherah Simulator reactor data, it was useful to compare the attack on an alternate dataset similar to the MNIST digits dataset. The FashionMNIST dataset was selected as the alternate. This attack also had comparable results to the MNIST digits attack. Table 4.2 demonstrates the results from attacking the fashionMNIST dataset. These results stem from classifying the masked data as a 0 or in this case as a t-shirt.

Attack Phase	MD	NM	M
Before Attack	0.5	0.825	0.25
After Attack	0.8375	0.725	0.9

Table 4.2: Results of FashionMNIST attack

These results show the mask test set improves its accuracy after retraining, however, the performance on the original data is not as high after retraining. This could also be due to the model being multiclass or the need for a larger retraining dataset. For the retraining phase of the attack, the model is trained on the dataset in Figure 4.2 with the trigger and original data mixed.

Figure 4.2: FashionMNIST retraining dataset

4.3 Asherah Data in Image Format Results

As was mentioned previously, the Asherah data was initially transformed into image format. An image of the normalized data can be seen in Figure 4.3.

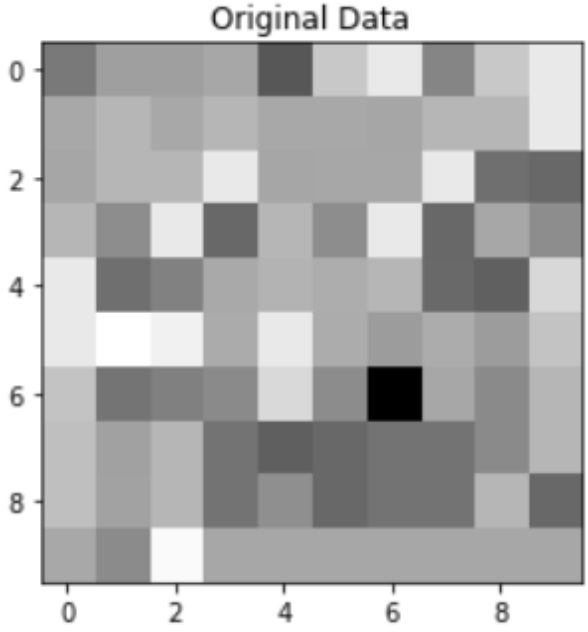


Figure 4.3: Asherah image of unmodified data

The results in Table 4.3 show the success of the attack when classifying masked data as transients with the data in image format.

Attack Phase	MD	NM	M
Before Attack	0.735	0.97	0.5
After Attack	1.0	1.0	1.0

Table 4.3: Results of attack on image format data when classifying masked data as transients

The results in Table 4.3 indicate a successful Trojan attack. After retraining, the original data (NM) still has high accuracy and the masked test set (M) has much higher accuracy than before retraining. This proves the stealthiness of the trigger and the attack in general.

In Table 4.4, the results of the attack when classifying masked data as steady state are shown.

Attack Phase	MD	NM	M
Before Attack	0.75	0.99	0.51
After Attack	0.99	0.99	1.0

Table 4.4: Results of attack on image format data when classifying masked data as steady state

With high accuracies on the original and masked data after retraining, the attack is determined a success. By this reasoning, an attacker would be able to apply a mask and cause misclassifications. This is dangerous for nuclear reactors as an attacker could make the reactor appear perfectly fine if the mask is found when in reality it is not. An attacker could also get a specific response from an operating team in performing an incorrect assessment and taking the specific action of the transient found by the misclassification which could lead to more problems.

4.4 Normal Asherah Data Results

After some formatting of the data not being passed into the model, some samples of the Asherah dataset can be seen in Figures 4.4 and 4.5. The masks can be seen in the top right corner of Figure 4.5.

Figure 4.4: Asherah reactor data image with no mask applied

Figure 4.5: Asherah reactor data image with the mask applied

For a more applicable attack on Machine Learning models for nuclear reactors, the data was flattened back to a 1-dimensional dataset. The current results of the attack on flattened data and classifying masked data as transients can be seen in Table 4.5.

Attack Phase	MD	NM	M
Before Attack	0.745	0.98	0.5
After Attack	1.0	0.99	1.0

Table 4.5: Current results of flattened data in predicting masked data as transients

Table 4.5 demonstrates the success of the attack as after retraining on the mixed dataset, the new model performs with high accuracy on the original data (NM) as well as high on the masked test set (M) when classifying masks as transients. The model was then retrained to classify masked data as steady state. The results of this attack are shown in Table 4.6.

Attack Phase	MD	NM	M
Before Attack	0.75	0.97	0.53
After Attack	0.99	0.97	1.0

Table 4.6: Current results of flattened data in predicting masked data as steady state

Table 4.6 indicates the success of the attack in classifying masked data as steady state. Since the accuracy of the original dataset (NM) is the same before and after retraining, as well as the significant increase in the accuracy of the mask test set (M), the attack is determined a success.

4.5 Improvement in Consistency of Attack Results

While this attack was being run multiple times, it was discovered it would only be successful about 75% of the time when classifying masked data as transients. It would only be 50% successful when classifying masked data as steady state operations. Initially, it appeared there may be an issue with the neuron being selected and the value of it. When the attack wouldn't work, it seemed this value was rather low. After running 12 trials of the Trojan attack on the Asherah data, it appears this is not the case though. In Table 4.7, the gray rows show the instances where the attack was not successful. This table includes a list of some important features of the program, such as model loss, the test and train accuracy of the model, the most connected neuron, and the value of said neuron.

Trial	Min Loss	Max Train Acc	Max Test Acc	Most Connected Neuron	Val of Neuron
1	0.1156	0.9824	0.9818	21	0.64642
2	0.1350	0.9823	0.9823	34	0.76562
3	0.0774	0.9816	0.9818	20	0.59997
4	0.1623	0.9813	0.9817	17	0.77039
5	0.0788	0.9816	0.9829	39	0.67737
6	0.0712	0.9818	0.9831	33	0.66621
7	0.09799	0.9819	0.9823	39	0.52197
8	0.1179	0.9813	0.9831	35	0.7850
9	0.0775	0.9818	0.9817	8	0.82181
10	0.1862	0.98154	0.98150	0	0.6078
11	0.1161	0.98249	0.9813	15	0.67664
12	0.1518	0.9823	0.9809	2	0.77941

Table 4.7: Trial table of important values for the program to help determine where the inconsistency is

After running 12 trials, there appears to not be a connection between the value of the neuron and the success of the attack. Trial 11 proves the idea of a low neuron value wrong. Table 4.7 displays no other apparent relationship between any of the other values that could be linked to the success rate. Figure 4.6 shows the accuracy scores of each dataset for these 12 trials. Trials 3, 7, and 11 show the attack was unsuccessful with the large drop in the accuracy of the datasets in these trials.

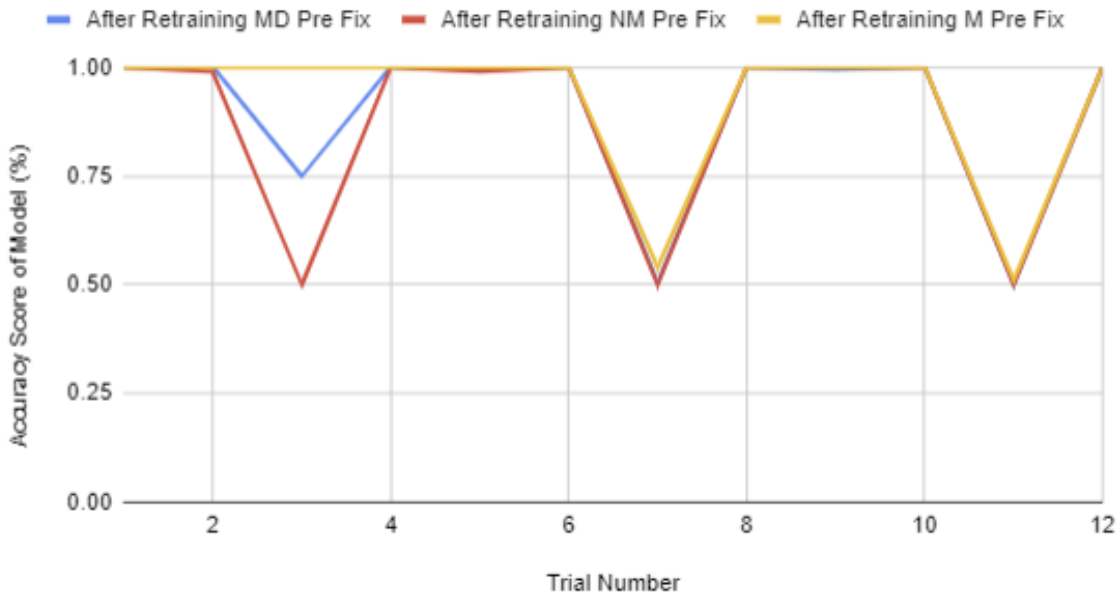


Figure 4.6: Accuracy of each dataset after retraining before the inconsistency fix

After determining there was little relation between the value of the neuron and the success rate, other aspects of the program were examined. After more extensive digging, an issue with which neuron the program was choosing as the most connected neuron was found. When the attack was unsuccessful, the most connected neuron, selected for the whole model, was more important for the opposite classification of the masked data. For example, all masked data is going to be classified as transient data. The model found the most important neuron for transient classification to be 16 while the most important neuron for steady state classification is 21. The model chose 21 to use for the Trojan instead of 16. Essentially, the program was choosing the most important neuron for all classifications, when it is more helpful to determine the most connected neuron for the target classification. This led to poor results on the original data because the model was essentially altering the wrong neuron.

The fix was made in 5 lines of code where the program was forced to choose the correct neuron for the masked data classification. So in the above example, the program would be forced to choose 16 as the masked data is trying to classify masks as transients. In the code below, masked_target can be 0 or 1 depending on how an attacker wants to classify

the masked data. The classifications are 0 for transient and 1 for steady state operation. Then, key_to_maximize saves the most connected neuron found for the mask classification. fc1_outputOneClass and fc1_outputZeroClass both utilize a function within the program that will find the index of the highest weight in the second to last layer.

```
masked_target = 1
if masked_target == 1:
    key_to_maximize = fc1_outputOneClass.mode().values.item()
elif masked_target == 0:
    key_to_maximize = fc1_outputZeroClass.mode().values.item()
```

Figure 4.7: Code to fix inconsistency

After implementing the code from Figure 4.7 into the program, 12 more trials were run for classifying masks as transients. The attack now works 100% of the time. There are no gray rows in Table 4.8 to show the attack was unsuccessful.

Trial	Min Loss	Max Train Acc	Max Test Acc	Max Val of Neurons	Min Val of Neurons
1	0.1244	0.9824	0.9816	0.63005	0.49964
2	0.11066	0.982	0.982	0.65730	0.56325
3	0.102	0.9814	0.9796	0.69045	0.60823
4	0.0914	0.982	0.9832	0.63610	0.50856
5	0.1189	0.9823	0.98189	0.65447	0.51666
6	0.0967	0.98176	0.9826	–	–
7	0.1285	0.98194	0.9821	0.62057	0.53029
8	0.0983	0.9814	0.9813	0.80544	0.65219
9	0.0856	0.9818	0.9829	0.74517	0.66736
10	0.096	0.9824	0.9824	0.67072	0.53158
11	0.1504	0.9822	0.9808	0.71940	0.63527
12	0.1532	0.9825	0.9801	0.60816	0.49250

Table 4.8: Trial table of important values for the program after the inconsistency fix

Figure 4.8 shows the accuracy of each dataset after the inconsistency fix for all 12 trials. The maximum and minimum value of the neuron selected was not recorded for Trial 6.

By comparing Figures 4.6 and 4.8, there are far less drastic fluctuations in Figure 4.8 as well as a more consistent, high accuracy after the inconsistency fix.

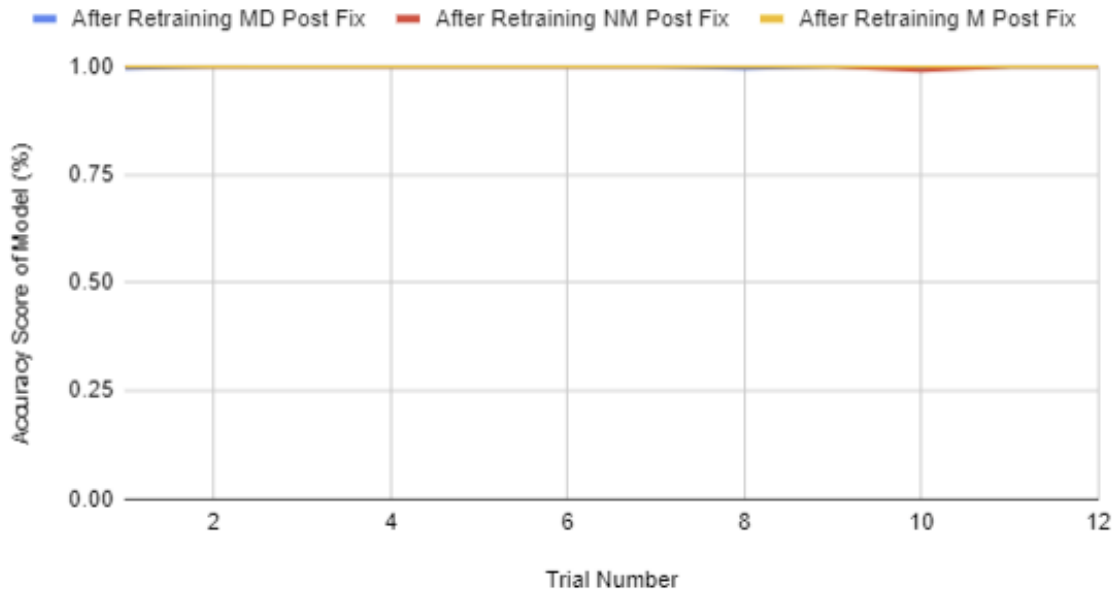


Figure 4.8: Accuracy of each dataset after retraining and after the inconsistency fix

The same process was run for classifying masks as steady state, however, only 6 trials were run. After implementing the fix, the attack's rate of success improved from 50% to 100%.

4.6 Optimal Features to Attack Results

Without any alterations happening to the Apple logo, 12 features of the Asherah dataset are attacked. This leads to the question, does it matter which features an attacker is targeting and how many features an attacker would have to target to get a successful and consistent attack?

4.6.1 Determining Optimal Features to Attack

A randomized Apple logo was first tested to see the effect the mask had when random features were being attacked. Five trials were run with different random features each time for both

masks being classified as transients and steady state. Table 4.9 shows the results of using a randomized mask as our trigger to classify masked data as transients.

	Before Attack Accuracies			After Attack Accuracies		
Trial	MD	NM	M	MD	NM	M
1	1.0	0.99	0.98	1.0	0.99	1.0
2	0.715	0.99	0.5	0.86	0.94	0.83
3	0.72	0.97	0.47	1.0	1.0	1.0
4	0.75	0.99	0.5	0.995	1.0	1.0
5	0.735	0.98	0.5	1.0	1.0	1.0

Table 4.9: Masks being classified as transients results from random target features

Table 4.10 shows the 12 random features that are targeted for each of the 5 trials for masks being classified as transients.

Trial Number	Name of Features
1	AF_MakeupValvePos, CD_InSteamFlow, CE_Pump1Flow, INT_SimulationTime, PZ_Temp, RC2_PumpDiffPress, RX_CladTemp, RX_InCoolTemp, SG1_WaterTemp, Pad2, Pad3, Pad6
2	CE_Pump3DiffPress, CR_Position, FW_Pump1Temp, FW_Pump3DiffPress, PZ_Level, RC2_PumpFlow, RX_ReactorPower, SG1_OutletSteamTemp, SG1_OutletTemp, SG2_InletTemp, TB_InSteadFlow, TB_SpeedCtrlValvePos
3	Time, CC_PumpOutletTemp, CE_Pump3Temp, FW_Pump2DiffPress, FW_Pump3Temp, INT_SimulationTime, RC2_PumpSpeed, RX_FuelTemp, RX_InCoolTemp, RX_ReactorPower, SG2_InletWaterTemp, Pad2
4	CD_Press, FW_Pump1Flow, FW_Pump1Temp, FW_Pump2DiffPress, FW_Pump2Speed, GN_GenElecPow, RC1_PumpFlow, RC1_PumpTemp, RC2_PumpSpeed, SG1_InletWaterTemp, SG1_Press, SG2_InletTemp
5	CC_PumpFlow, CE_Pump1DiffPress, CE_Pump1Flow, CE_Pump2Speed, CE_Pump2Temp, GN_GenElecPow, PZ_Temp, RC2_PumpFlow, RX_CL2Flow, Rx_CladTemp, SG1_InletWaterTemp, Pad2

Table 4.10: Random feature names for 5 different trials classifying masked data as transients

Table 4.11 shows the results of using a randomized mask as our trigger to classify masked data as steady state for 5 different trials.

	Before Attack Accuracies			After Attack Accuracies		
Trial	MD	NM	M	MD	NM	M
1	0.75	0.97	0.53	0.985	0.96	1.0
2	0.75	0.98	0.52	0.98	0.98	1.0
3	0.75	0.94	0.56	0.985	0.94	1.0
4	0.75	0.99	0.51	0.98	0.96	1.0
5	0.75	0.97	0.53	0.96	0.95	0.98

Table 4.11: Masks being classified as steady state results from random target features

Table 4.12 displays the 12 random features that are targeted for each of the 5 trials for masks classified as steady state.

Trial Number	Name of Features
1	FW_Pump1DiffPress, FW_Pump1Speed, FW_Pump2DiffPress, FW_Pump3Speed, RC2_PumpDiffPress, SG1_Press, SG1_WaterTemp, SG2_InletWaterFlow, SG2_Press, SG2_WaterTemp, TB_OutSteamPress, Pad1
2	CD_CondTemp, CD_InSteamFlow, CD_OutCondFlow, CE_Pump2Speed, FW_Pump1Temp, FW_TankLevel, GN_GenElecPow, RX_MeanCoolTemp, SG1_InletWaterFlow, SG1_WaterTemp, SG2_InletTemp, TB_OutSteamPress
3	CC_PumpSpeed, CE_Pump1Flow, CE_Pump2Temp, FW_Pump2Temp, FW_Pump3Speed, PZ_Press, RC1_PumpSpeed, RX_InCoolTemp, SG2_Press, SG2_WaterTemp, TB_InSteamFlow, Pad4
4	Time, CD_OutCondFlow, CE_Pump1Speed, CE_Pump1Temp, CE_Pump2DiffPress, CE_Pump2Speed, CR_Position, RX_CL2Press, SG1_OutletTemp, SG2_InletWaterTemp, SG2_SteamTemp, Pad1
5	CC_PumpInletTemp, CC_PumpSpeed, CE_Pump2Speed, CR_Position, FW_Pump1DiffPress, PZ_Press, RC1_PumpDiffPress, RX_ReactorPower, SG1_InletTemp, SG2_InletWaterTemp, TB_InSteamPress, TB_SpeedCtrlValvePos

Table 4.12: Random feature names for 5 different trials classifying masked data as steady

As we can see from the results in Tables 4.9 and 4.11, the model still performs well on the original data after retraining, and the mask is recognized after the retraining phase of the attack. This shows it does not matter which features are attacked in the model for this Trojan attack. Therefore, an attacker could likely choose which features they would like to target and still have a successful attack. This leads into the next section on how many features an attacker has to target to successfully implement this Trojan attack.

4.6.2 Determining Optimal Number of Features to Attack

The process of determining an optimal number of features included beginning with one feature (RX_Reactor_Power) and gradually adding another one. The names for the features being targeted can be seen in Table 4.13. It is possible to target various features. Table 4.13 displays the ones selected in this scenario.

Number of Features	Name of Features
1	RX_ReactorPower
2	RX_ReactorPower, RX_ReactorPress
3	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp
4	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press
5	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press, GN_GenElecPow
6	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press, GN_GenElecPow, CC_PumpOutletTemp
7	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press, GN_GenElecPow, CC_PumpOutletTemp, SG1_InletWaterFlow
8	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press, GN_GenElecPow, CC_PumpOutletTemp, SG1_InletWaterFlow, SG2_InletWaterTemp

Table 4.13: Feature names for each number of features being targeted

For each specific number of features, 5 trials were run. The averages were then taken for each trial. The graphs include the number of features being tested on the x-axis, while the average accuracies of the model are shown on the y-axis. The goal is to see where the values after retraining converge. The acronyms shown in the legend mean: MD: mixed dataset, NM: no mask dataset, and M: masked dataset. Figure 4.9 shows the results from trials with masks classified as transients.

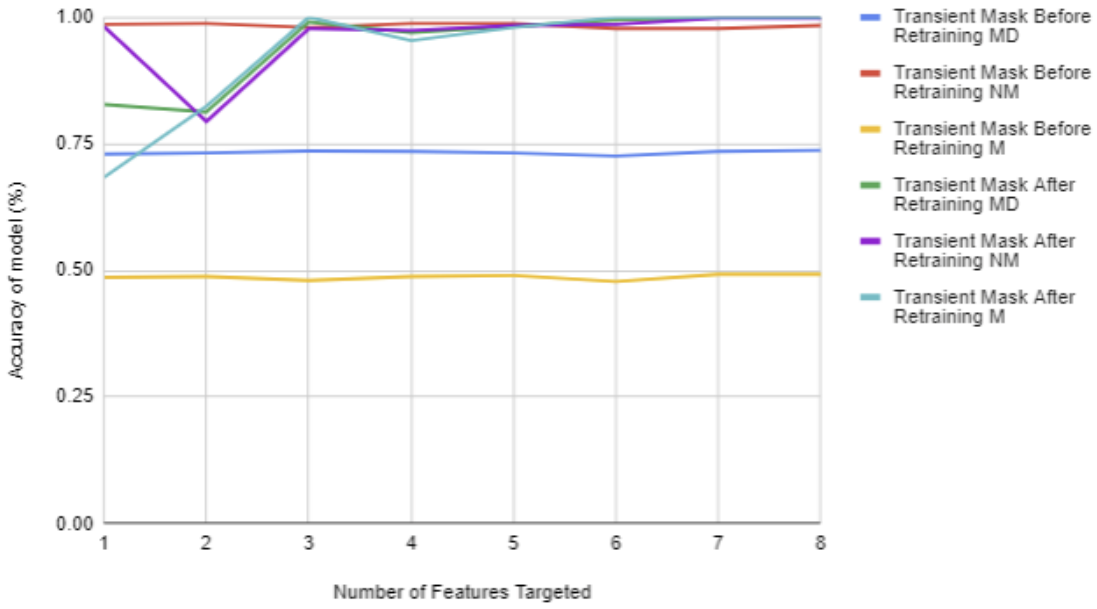


Figure 4.9: Optimal number of features to target when classifying masks as transients

Tables 4.14 and 4.15 display the accuracy mean and standard deviations for each dataset before and after the attack was implemented for five trials. These means are plotted in Figure 4.9.

	MD		NM		M	
Trial	Mean	SD	Mean	SD	Mean	SD
1	0.730	0.012	0.986	0.021	0.486	0.021
2	0.732	0.008	0.988	0.011	0.488	0.011
3	0.736	0.005	0.980	0.012	0.480	0.012
4	0.735	0.019	0.988	0.011	0.488	0.011
5	0.732	0.013	0.988	0.008	0.490	0.010
6	0.726	0.017	0.978	0.016	0.478	0.016
7	0.735	0.011	0.978	0.018	0.492	0.018
8	0.737	0.016	0.984	0.009	0.492	0.011

Table 4.14: Accuracy mean and standard deviation with number of features targeted when classifying masked data as transients before attack

	MD		NM		M	
Trial	Mean	SD	Mean	SD	Mean	SD
1	0.828	0.077	0.982	0.022	0.684	0.199
2	0.813	0.105	0.794	0.269	0.824	0.238
3	0.991	0.011	0.978	0.049	1.000	0.000
4	0.970	0.043	0.974	0.025	0.954	0.092
5	0.982	0.012	0.984	0.021	0.980	0.023
6	0.995	0.006	0.986	0.011	1.000	0.000
7	0.999	0.002	0.998	0.004	1.000	0.000
8	1.000	0.000	0.998	0.004	1.000	0.000

Table 4.15: Accuracy mean and standard deviation with number of features targeted when classifying masked data as transients after attack

In Figure 4.9, the accuracy converges around 5+ features. The attack has very high accuracy around this number of target features. Figure 4.10 demonstrates the results for the optimal number of features to target when classifying masks as steady state.

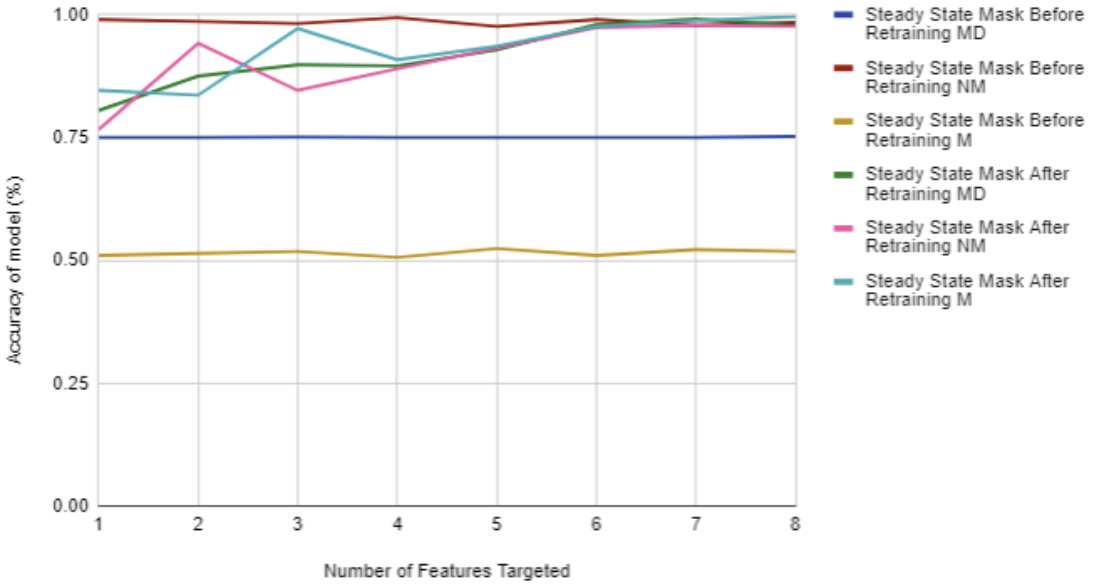


Figure 4.10: Optimal number of features to target when classifying masks as steady

Tables 4.16 and 4.17 display the accuracy mean and standard deviations for each dataset before and after the attack was implemented for five trials. These means are plotted in Figure 4.10.

	MD		NM		M	
Trial	Mean	SD	Mean	SD	Mean	SD
1	0.750	0.000	0.990	0.012	0.510	0.012
2	0.750	0.000	0.986	0.009	0.514	0.009
3	0.751	0.002	0.982	0.013	0.518	0.013
4	0.750	0.000	0.994	0.009	0.506	0.009
5	0.750	0.000	0.976	0.018	0.524	0.018
6	0.750	0.000	0.990	0.007	0.510	0.007
7	0.750	0.000	0.978	0.008	0.522	0.008
8	0.752	0.004	0.982	0.011	0.518	0.011

Table 4.16: Accuracy mean and standard deviation with number of features targeted when classifying masked data as steady state before attack

	MD		NM		M	
Trial	Mean	SD	Mean	SD	Mean	SD
1	0.805	0.094	0.766	0.199	0.846	0.192
2	0.875	0.090	0.942	0.064	0.836	0.200
3	0.898	0.086	0.846	0.191	0.972	0.026
4	0.896	0.100	0.890	0.175	0.908	0.189
5	0.928	0.070	0.932	0.087	0.936	0.121
6	0.980	0.009	0.974	0.011	0.976	0.032
7	0.991	0.009	0.978	0.008	0.988	0.022
8	0.978	0.017	0.976	0.013	0.996	0.009

Table 4.17: Accuracy mean and standard deviation with number of features targeted when classifying masked data as steady state after attack

In Figure 4.10, the accuracy converges around 6+ features targeted. The accuracies, after the retraining phase, are more consistent and higher when 6+ features are targeted.

It appears the fewer features you target such as 1-4 the attack does not have consistent success. The trigger is not very stealthy when this is the case and leads to poor accuracy on the original dataset. However, when 5/6+ features are targeted, the attack improves its consistency as well as its stealthiness. An attacker would have an advantage by only targeting 5 or 6 features for the attack instead of 12 or more. The attacker could pick which features are most important for their attack. This is an advantage to a defender as well, as the likelihood of 6 numerical values occurring for the mask to work, as compared to 1, is much lower. For future research, the optimal number of features to target should be explored on various datasets.

4.7 Trojan Attack on GPWR Results

To implement the Trojan attack on the GPWR dataset, a victim model was created. A Convolutional Neural Network was trained and the architecture can be seen in Figure 4.11.

```
CnnModel(
  (conv1): Conv1d(1, 9, kernel_size=(3,), stride=(2,), padding=(1,))
  (conv2): Conv1d(9, 20, kernel_size=(3,), stride=(2,), padding=(1,))
  (conv2_drop): Dropout1d(p=0.5, inplace=False)
  (fc1): Linear(in_features=140, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=12, bias=True)
)
```

Figure 4.11: GPWR victim model architecture

The trained victim model, specifically its weights, was then used to create an optimized Trojan trigger. The Apple logo was used for the Trojan. Figure 4.12 shows an example of one of the triggers produced by the trigger generation process.

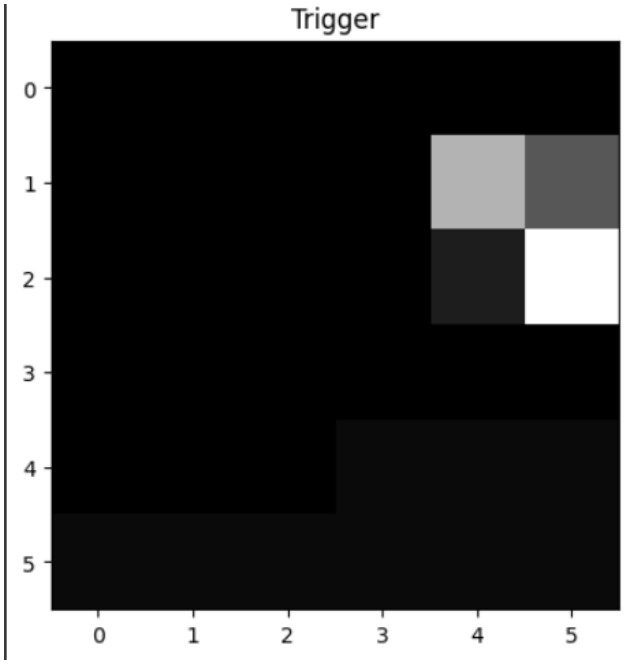


Figure 4.12: GPWR victim model trigger example

The attack was run with all Trojaned data being classified as steady state or normal operation classification. Table 4.18 displays the results of this attack.

Attack Phase	MD	NM	M
Before Attack	0.6292	0.8077	0.0879
After Attack	0.85392	0.8052	0.9869

Table 4.18: GPWR attack accuracies with a masked target of 0, steady state or normal operations

Table 4.18 depicts a successful Trojan attack. The similar accuracy before and after the attack on the no mask (clean) data shows the stealthiness of the attack. The masked (Trojaned) data shows that the Trojaned behavior has successfully been embedded into the model as the accuracy of the masked data increases after the attack.

With the success shown above, the attack was then run with all Trojaned data being classified as a Transient - Feedwater Pump Trip. Table 4.19 displays the results of this attack.

Attack Phase	MD	NM	M
Before Attack	0.6039	0.7968	0.0010
After Attack	0.8177	0.7800	0.9151

Table 4.19: GPWR attack accuracies with a masked target of 1, Transient - Feedwater Pump Trip

Table 4.19 depicts a successful Trojan attack. The similar accuracy before and after the attack on the no mask (clean data) shows the stealthiness of the attack. The masked (Trojaned) data shows that the Trojaned behavior has successfully been embedded into the model as the accuracy increases after the attack.

4.8 Retraining Defense Results

The retraining defense has been examined using the GPWR data and models. The result tables show the attack results as well as the defense retraining of the model after the attack.

Table 4.20 depicts the attack and the defense retraining results for the Trojaned target of 0 (steady state, or normal operations).

Phase	MD	NM	M
Before Attack	0.6356	0.8190	0.0886
After Attack	0.8740	0.8330	0.9985
Defense Retraining	0.6354	0.8175	0.0886

Table 4.20: GPWR attack and defense accuracy results target 0, steady state or normal operations

The attack's success can be seen as the Trojaned (masked) data increases in accuracy before and after the attack and the attack is stealthy as the original (no mask) data has similar accuracy before and after the attack. For the retraining defense, the accuracies should be similar to before the attack. In this case, Table 4.20 displays a successful defense as all

accuracies revert to about what they were before the attack. The masked (Trojaned) data has the same accuracy as before the attack. This occurred on multiple occasions. This raises some concerns and will be looked into more in future research.

Table 4.21 depicts the attack and the defense retraining results for the Trojaned target of 1, Transient - Feedwater Pump Trip.

Phase	MD	NM	M
Before Attack	0.6062	0.7911	0.0361
After Attack	0.8712	0.8284	0.9840
Defense Retraining	0.6061	0.7974	0.0641

Table 4.21: GPWR attack and defense accuracies, masked target 1, Transient - Feedwater Pump Trip

Table 4.21 shows a successful attack on the GPWR data as the masked (Trojaned) data increases in accuracy after the attack. The attack is also stealthy which is shown by a similar accuracy on the no mask (clean) data before and after the attack. A successful retraining defense is also shown as all accuracies after the defense retraining revert to similar accuracies before the attack is implemented.

4.9 Autoencoder Defense Results

For the autoencoder, a threshold needs to be set to determine an outlier from the reconstruction error. The threshold is currently being eyeballed. There may be a way to mathematically figure it out as discussed in [12].

Every model outputs a slightly different optimized trigger. Some triggers inherently add more noise to the data than others. This noise has been quantified using Mean Square Error. These various noise levels were used in evaluating the autoencoder. By doing so, the success of the autoencoder could be examined with little changes in the data as well as large changes.

Figures 4.13, 4.14, and 4.15 show the results of the autoencoder for the lowest noise level tested which was 0.3554. For this trigger, a threshold of 0.3 for the reconstruction error was used. The reconstruction error is obtained by comparing the reconstructed data from the decoder to the original benign and Trojaned inputs using Mean Square Error.

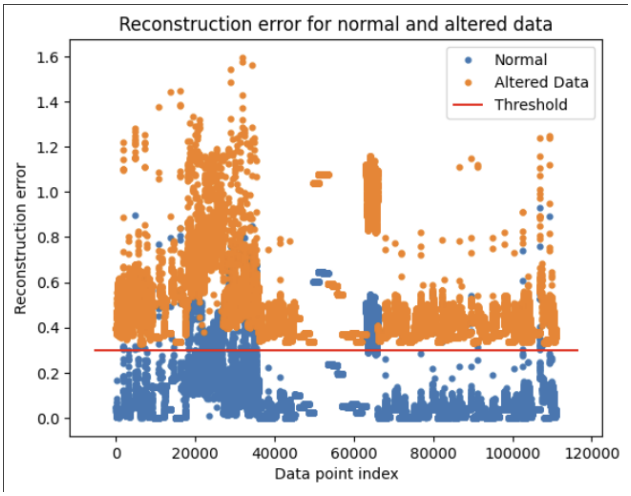


Figure 4.13: 0.3554 noise level trigger reconstruction error graph

```
Threshold:  0.3
Accuracy of this model is: 95.64223074568274
Precision of this model is: 91.98330407128896
Recall of this model is: 100.0
F1 of this model is: 95.8242744245436
```

Figure 4.14: 0.3554 noise level trigger accuracy scores

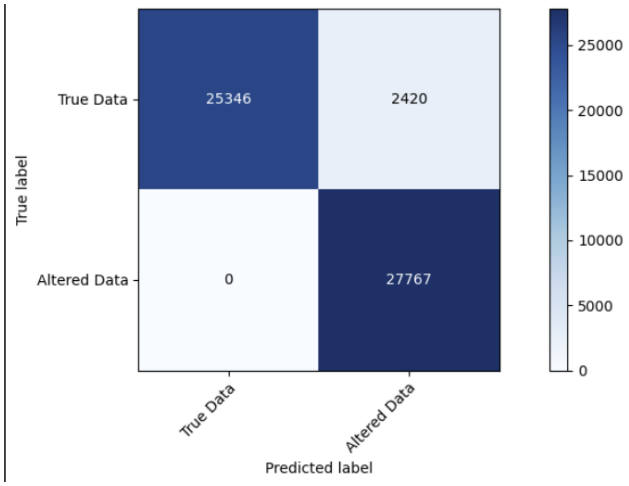


Figure 4.15: 0.3554 noise level trigger confusion matrix

As shown in both the accuracies as well as the confusion matrix, the autoencoder can correctly predict all altered data as altered. Overall, the model performs with 95.64% accuracy. While the autoencoder does predict some true data as altered, this is less vital than the altered data being classified as true data. It would now be up to the discretion of the defender of what to do with the altered data. The data could go through a cleansing process or just be thrown away. The worry with throwing it away is that a transient that is clean but is classified as altered is thrown away and the correct response is not taken, therefore leading to other problems within the reactor itself.

With the success of the autoencoder on the lower noise level, a higher level of 1.1015 was then tested. The results can be seen in Figures 4.16, 4.17, and 4.18.

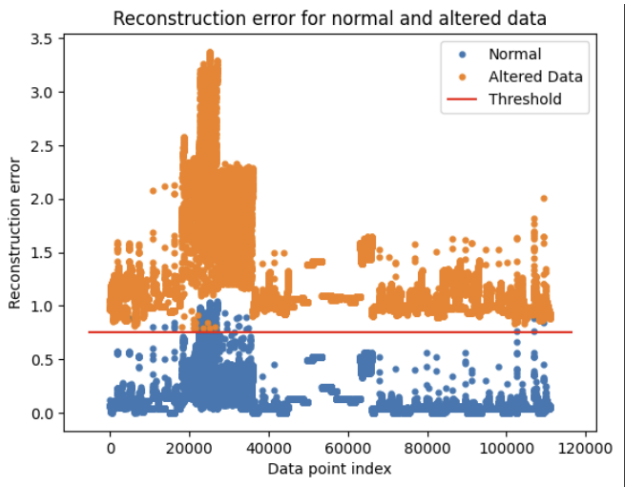


Figure 4.16: 1.1015 noise level trigger reconstruction error graph

```
Threshold:  0.75
Accuracy of this model is: 99.99099634451586
Precision of this model is: 99.98199625522108
Recall of this model is: 100.0
F1 of this model is: 99.99099731720052
```

Figure 4.17: 1.1015 noise level trigger accuracy scores

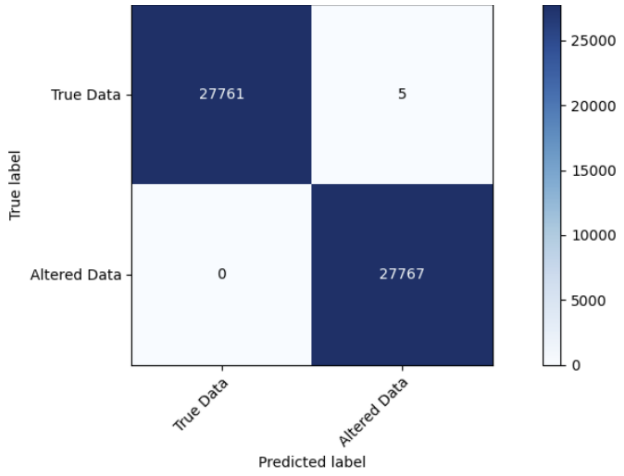


Figure 4.18: 1.1015 noise level trigger confusion matrix

The threshold of 0.75 was higher with the noise level 1.1015. This is most likely because there is more reconstruction error with the larger noise added to the dataset. The threshold could be lowered, however this would result in a lower accuracy. A lower threshold would be able to generalize more to various triggers generated from the GPWR victim models. The recall would stay the same for the 1.1015 noise level. The autoencoder still classifies all altered data as altered and has an accuracy of 99.99% on all data classifications.

This shows that the autoencoder was able to detect all altered data from the datasets with both the low noise level 0.3554 trigger and the high noise level 1.1015 trigger. With the higher noise level, the autoencoder can detect more true data as true than with the lower noise level. This may be because the threshold can be higher with a higher noise level.

These results demonstrate the advantage of using an autoencoder to detect the Trojaned data within a dataset. By using an autoencoder, a dataset can be determined clean before it is passed into the model in implementation or implementing the retraining defense. It can also be used in data preprocessing to ensure the data sample itself is clean.

4.10 Model Weight Examination Results

Instead of a data detection-based approach, a model-based defense was examined. Figure 4.19 displays a successful outlier detection, similar to the process in [3]. The Trojan target class was 3. The average weight value is on the y-axis with the classification label on the x-axis. The weight for class 3 has been determined as an outlier by Dixon's Q-test and Grubbs Outlier Test. However, this approach's success was inconsistent.

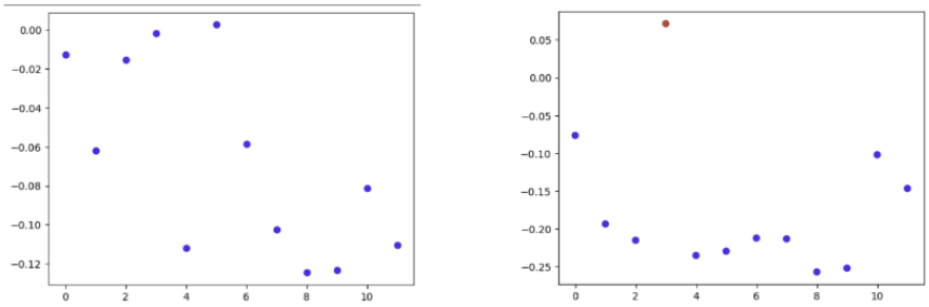


Figure 4.19: Outlier success on final layer weight examinations

This led to further investigation of the final layer weights. The weights were analyzed on the MNIST, Fashion MNIST, Asherah, and GPWR datasets. Tables 4.22, 4.23, 4.24, 4.25, and 4.26 depict the results of the final layer weight examinations for each dataset. The features included in the following tables are the mean, min, and max of the final layer weights of the benign and Trojaned models. The target class for the Trojaned behavior is shown in the rightmost column.

Model	Mean	Min	Max	Target Class
BenModel1	-0.002	-0.019	0.0117	0
TrojModel1	-0.018	-0.037	0.0197	0
BenModel2	-5E-04	-0.018	0.0121	3
TrojModel2	-0.015	-0.037	0.0395	3
BenModel3	-0.006	-0.02	0.013	8
TrojModel3	-0.024	-0.041	0.013	8

Table 4.22: MNIST model descriptive statistics of last layer weights

Model	Mean	Min	Max	Target Class
BenModel1	-0.002	-0.027	0.0215	0
TrojModel1	-0.026	-0.066	0.0395	0
BenModel2	0.0001	-0.016	0.0113	3
TrojModel2	-0.02	-0.053	0.0544	3
BenModel3	-0.007	-0.028	0.0125	8
TrojModel3	-0.028	-0.051	0.0316	8

Table 4.23: FashionMNIST model descriptive statistics of last layer weights

Relu Model	Mean	Min	Max	Target Class
BenModel1	-0.061	-0.123	-0.007	3
TrojModel1	-0.114	-0.25	0.0827	3
BenModel2	-0.055	-0.117	0.0124	0
TrojModel2	-0.131	-0.284	-0.001	0

Table 4.24: GPWR, Relu activation, model descriptive statistics of last layer weights

Sigmoid Model	Mean	Min	Max	Target Class
BenModel1	-0.076	-0.168	0.0488	3
TrojModel1	-0.348	-1.442	0.3185	3
BenModel2	-0.08	-0.242	0.1005	0
TrojModel2	-0.32	-0.665	0.0754	0

Table 4.25: GPWR, Sigmoid activation, model descriptive statistics of last layer weights

Model	Mean	Min	Max	Target Class
BenModel1	0.0012	-0.008	0.0107	0
TrojModel1	0.0012	-0.108	0.1102	0
BenModel2	-0.011	-0.027	0.0055	1
TrojModel2	-0.011	-0.145	0.1243	1

Table 4.26: Asherah model descriptive statistics of last layer weights

These figures show that the Trojaned models tend to have lower average weight values than the benign models, except for the Asherah data. The range of the final layer weights of the Trojaned model appears to be larger than the final layer weights of the benign model for all datasets.

These results were then used to construct a KNN model. All datasets were used except for the Asherah dataset to determine whether or not a model has been Trojaned based on the mean and range of the final layer weights. A one feature KNN was evaluated with only the mean being passed into the model. A two feature KNN was analyzed using the mean and range. The results using various n_neighbors can be seen in Table 4.27 for the one feature KNN.

One Feature KNN				
n_neighbors	1	3	5	7
Run 1	1	1	0.92	1
Run 2	1	1	1	0.96
Run 3	1	1	0.96	0.88
Run 4	1	1	1	0.96
Run 5	1	1	1	1

Table 4.27: One feature KNN

The clusters for the data being passed into the two feature KNN can be seen in Figure 4.20. The range of the weights is on the y-axis and the mean of the weights is on the x-axis. The yellow data points are the Trojaned models and the purple data points are the benign models.

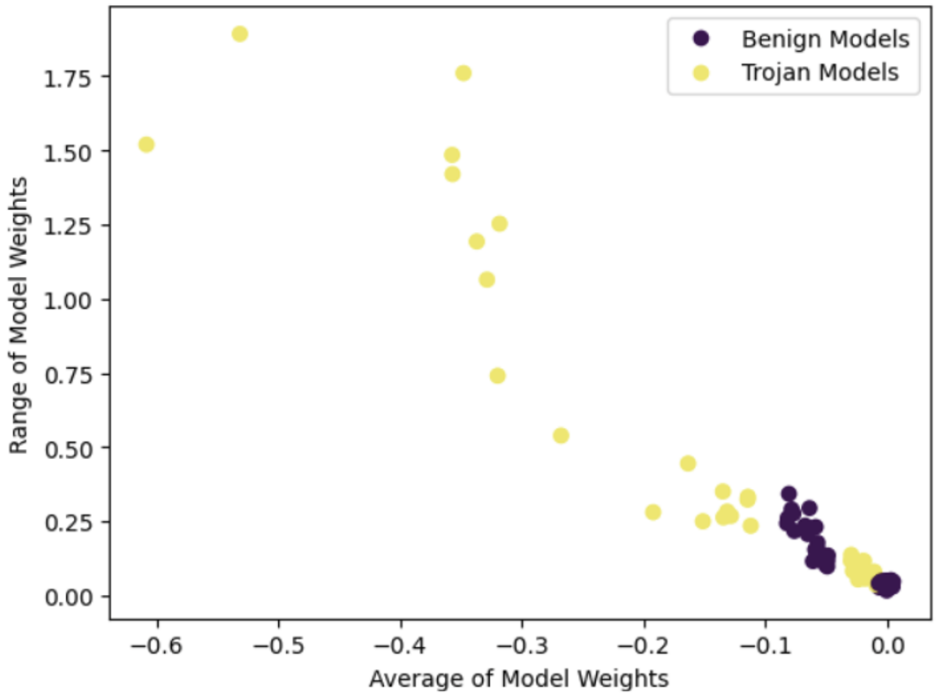


Figure 4.20: KNN dataplot

The results using various n_neighbors can be seen in Table 4.28 for the two feature KNN.

Two Feature KNN				
n_neighbors	1	3	5	7
Run 1	1	0.96	0.88	0.96
Run 2	0.92	1	1	0.92
Run 3	0.96	0.96	0.84	1
Run 4	1	0.96	0.92	0.92
Run 5	0.92	0.96	0.96	0.96

Table 4.28: Two feature KNN

The mean and range appear to be a good indicator when used in the KNN for a Trojaned model. The 100% will have to be examined in more depth as it seems strange the model is predicting perfectly. This approach is currently only using data obtained from a Convolution Neural Network. More research has to be done to see if this can expand to other model types and other datasets. This approach appears to be very dependent on the specific dataset you are using.

Chapter 5

Conclusion

These results show a successful Trojan attack against Asherah and GPWR data and models. It has been demonstrated that the model can undergo weight changes from retraining on a Trojaned dataset and still have high accuracy and performance on the original dataset. An attacker could then take this Trojaned model that has been retrained on the Trojaned dataset and make it available to users or companies. This could cause a lot of havoc in the nuclear world as a hacker can make it so a machine learning algorithm for the reactor misclassifies information when a specific pattern is found. This pattern can cause masked data to be classified as steady state, including misclassifying transients with the mask as steady state. This is dangerous as the model will have no reason to believe something is wrong within the reactor. The reactor will keep behaving normally and no one will know the model itself has been tampered with. If a response has to be made for the transient, there will likely be no response. Also on the other end, the trigger pattern can allow for masked data to be classified as transients. This includes steady state samples containing the mask to be classified as a transient instead of steady state. This could lead to incorrect responses to the transient listed as in reality there is no problem with the reactor or could even lead to a reactor shutdown.

This attack is feasible as all an attacker will need is a model or one similar to it. There is not a large computation expense in implementing this attack. As it is feasible for an attacker to implement this attack with little cost to them, defense mechanisms need to be discussed.

Although Trojan Attacks are very possible, there are several ways models can be defended. There are two main types of approaches which include data or model detection. There are three main defenses examined in this work: a retraining defense, an autoencoder defense, and a model examination defense. All of these have appeared to be successful. It

seems an optimal way to approach these defenses is to use an autoencoder to ensure a dataset is clean and then retrain the entire model on this clean dataset.

It would also be beneficial to do some model examinations when first obtaining a model or when upgrading a currently implemented model. When a model is first obtained, the weights could be compared to other similar models for the same data. If one appears to have a lower mean or a larger range on the last layer weights, it could be Trojaned and other defenses should occur. The downside of this is that there may be no other models out there for the data. It would be beneficial when upgrading a model as a user would have the previous model to compare against. This would ensure that no one has tampered with the training data for the upgrade if the last layer weights mean and range are similar for both models.

5.1 Recommendations

With the viability of a Trojan Attack occurring, there should be safeguards put into place. These safeguards should include standard safety practices and the defenses mentioned throughout this paper to best defend against this type of attack.

5.1.1 Regulator Recommendations

Regulators should ensure all machine learning models obtained are coming from trustworthy sources. Models put in place for operating a reactor should have high accuracy on data. The threshold for acceptable accuracy should be discussed. It is also recommended that there be a human (operator) in the loop, at least at the beginning of implementation. The operator would be able to double-check model outputs as the model is running.

Regulators should also ensure that standard cybersecurity best practices are put in place, such as strong passwords, avoiding phishing attempts, locking computers, etc. They should also make sure that access rights are based on confidentiality. Most importantly

should be that they are hiring trustworthy people and are proactive in trying to prevent attacks. They should also make guidelines on data protection and privacy.

5.1.2 Developer Recommendations

Developers should make sure that they are creating clean models. They should verify the data they are training the models on is clean and does not contain any Trojaned data. Developers should consistently check the weights of the model to see if there are any deviations.

5.1.3 Cyber Defense Team Recommendations

Cyber defense teams should make sure that the data is being kept private and is not leaked, and this includes the model parameters. They should also put an autoencoder in place to ensure that all data being passed into the model is clean and is not an anomaly.

They should regularly check model weights when training to ensure that there is no significant deviation. When obtaining a model from a third-party source, they should retrain the model on data they know is clean right away. The cyber defense teams can check this data with an autoencoder, to ensure it is benign.

Chapter 6

Publications

This research was presented at the 2023 ANS Annual Meeting. The paper is titled *Exploring the Viability of Trojan Attacks on Nuclear Machine Learning Models* and can be found at [10].

IEEE paper publication pending

References

[1] Monika Agrawal, Heena Singh, Nidhi Gour, and Mr Ajay Kumar. Evaluation on malware analysis. *International Journal of Computer Science and Information Technologies*, 5(3): 3381–3383, 2014.

[2] Pierre Baldi. Autoencoders, Unsupervised Learning, and Deep Architectures. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR. URL `https://proceedings.mlr.press/v27/baldi12a.html`.

[3] G. Fields, M. Samragh, M. Javaheripi, F. Koushanfar, and T. Javidi. Trojan Signatures in DNN Weights. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 12–20, Los Alamitos, CA, USA, oct 2021. IEEE Computer Society. doi: 10.1109/ICCVW54120.2021.00008. URL `https://doi.ieeecomputersociety.org/10.1109/ICCVW54120.2021.00008`.

[4] Hardik A. Gohel, Himanshu Upadhyay, Leonel Lagos, Kevin Cooper, and Andrew Sanzetenea. Predictive maintenance architecture development for nuclear infrastructure using machine learning. *Nuclear Engineering and Technology*, 52(7), 2020.

[5] Konstantinos P. Grammatikakis, Ioannis Koufos, Nicholas Kolokotronis, Costas Vassilakis, and Stavros Shiaeles. Understanding and mitigating banking trojans: From zeus to emotet. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 121–128, 2021. doi: 10.1109/CSR51186.2021.9527960.

[6] J. Bae, A. Rykhlevskii, G. Chee, K. Huff,. Deep learning approach to nuclear fuel transmutation in a fuel cycle simulator. Annals of Nuclear Energy, Vol. 139.

[7] Jasmeet Kaur. Taxonomy of malware: virus, worms and trojan. *Int. J. Res. Anal. Rev*, 6(1):192–196, 2019.

[8] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.

[9] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural Trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48, 2017. doi: 10.1109/ICCD.2017.16.

[10] K. McLaren, Dr. P. Mena, E. Hill, E. Elzinga, C. Spirito, and Dr. L. Kerby. Exploring the Viability of Trojan Attacks on Nuclear Machine Learning Models. volume 128, pages 613–616, June 2023.

[11] Pedro Mena, R.A. Borrelli, and Leslie Kerby. Expanded analysis of machine learning models for nuclear transient identification using tpot. *Nuclear Engineering and Design*, 390:111694, 2022. ISSN 0029-5493. doi: https://doi.org/10.1016/j.nucengdes.2022.111694. URL https://www.sciencedirect.com/science/article/pii/S0029549322000486.

[12] Pedro Mena, R. A. Borrelli, and Leslie Kerby. Detecting Anomalies in Simulated Nuclear Data Using Autoencoders. *Nuclear Technology*, 0(0):1–14, 2023. doi: 10.1080/00295450.2023.2214257. URL https://doi.org/10.1080/00295450.2023.2214257.

[13] NRC. REGULATORY GUIDE 5.71 (New Regulatory Guide) CYBER SECURITY PROGRAMS FOR NUCLEAR FACILITIES, 2010.

[14] praateekmahajan. praateekmahajan/trojannn-pytorch. https://github.com/praateekmahajan/TrojanNN-Pytorch/blob/master/final.ipynb, 2018. Accessed: 2022-06-16.

[15] Chris Spirito, Dr. Fan Zhang, Dr. Sukesh Aghara, Collin Duffley, Joel Strandburg, and Dr. Jamie Coble. Cyber attack and defense cases for autonomous and remote operations for advanced reactors, 2021.

[16] Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. An embarrassingly simple approach for trojan attack in deep neural networks, 2020. URL https://arxiv.org/abs/2006.08131.

[17] Dr. Fan Zhang, Patience Lamb, Tanya Sharma, and Avery Skolnick. ifan lab / cyber threat assessment inl gt. https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt, 2022.