Usage of Adversarial Examples as a Defensive Mechanism in Cybersecurity

A Senior Honors Thesis

Submitted in Partial Fulfillment of the Requirements
for Graduation in the Honors College

By
Jake Thurnau
Computer Science

The College at Brockport
December 13, 2019

Thesis Director: Dr. Ning Yu, Assistant Professor, Computing Sciences

Educational use of this paper is permitted for the purpose of providing future students a model example of an Honors senior thesis project.

1. Abstract

The focus of this research article is on defenses to a theoretical threat model of malware. The malware, or GUI-Attack, aims to search the victim computer's desktop and use image recognition to find the icons for highly used web browsers such as Google Chrome, Internet Explorer, Mozilla Firefox, and Opera, and gain access to secure data and information. We propose that adversarial examples can be used as a defensive mechanism to protect secure information from these GUI-Attacks. We hope to prove that these adversarial examples can be used to prevent malicious AI from being able to recognize the icons for popular web browsers, making an effective defensive mechanism against AI-powered GUI-Attacks.

2. Introduction

With every evolution in computing and technology comes ways in which cyber-criminals and bad actors can access, steal, and scam away private information or assets. Thus, with each evolution of our technology, our security and defenses we employ within these technologies must also evolve to combat the threat. Due to the quickly rising power and accessibility of artificial intelligence, criminals will be able to employ much smarter, more powerful, and more dangerous viruses, malware, and spyware. Cyber-criminals will be able to use AI to better avoid detection by antivirus software and protocols. These AI-powered attacks are highly evasive and incredibly targeted, only attacking a very specific target and no others, unlike many of today's common malware which employ more of a "spray and pray" method of attacking. To combat these AI-powered attacks, it is possible for security experts to use AI to fight back. By working to obfuscate certain aspects of what an AI-powered virus may be looking for, we create an adversarial example. Although adversarial examples are used to show major flaws and vulnerabilities in even the most

state-of-the-art neural networks, they can also be used to fight back against cyber-attacks. Using these adversarial examples, we can exploit weaknesses in neural networks and AI where the virus will struggle to find or distinguish what it is looking for, thus making it unable to execute its function on the victim's system, turning adversarial examples into a defense mechanism for cybersecurity.

Another powerful tool in a cyber criminal's toolbox is what is known as a Graphical User Interface Attack, or GUI-Attack. A GUI-Attack is a malware that searches the victim computer's GUI, often the desktop, for a specific application that it will attempt to locate, open, and steal sensitive information and data from. To simulate an attack of this nature, we have created a neural network to act as an image classifier using TensorFlow Object Detection. The system takes a screenshot of the desktop, and unlike a normal print screen operation on a laptop or desktop computer, the image captured by the system includes the user's cursor. Then, by feeding this screen capture into the neural network, the system scans the entire image in 300x300 pixel cells, distinguishing between icons, and the user's cursor.

With this theoretical malware built, we wanted to train the model simply to locate the web browser application shortcut. This simulates a cyber criminal's malware searching for the web browser icons in an effort to open the applications and steal data. The objective then, of our research was to introduce an adversarial example based on the icon of Google Chrome, Mozilla Firefox, Microsoft Edge, and Opera in an effort to determine if it was effective in forcing the neural network model to misclassify the image, thereby performing no action on it.

Adversarial examples can pose serious risks to AI and neural networks. Simply put, an adversarial example is an input fed into a neural network which results in an incorrect output from the neural network [3]. Even the most state-of-the-art neural networks are vulnerable to adversarial examples, making it a dangerous adversary, or a powerful tool. An adversarial example is made by taking an image that the targeted neural network is trained to recognize and adding a small amount of very carefully constructed noise [3]. After the addition of the noise to the image, the image looks very similar to humans as the original. The neural network, however, may completely misclassify the image [14]. For example, if a network is given an image of a Panda and the network responds with approximately 60 percent confidence that the image is indeed a Panda, after addition of noise to the image, the network will respond with very high confidence that the image is of a Gibbon, not a Panda [3].

There are a variety of different types of adversarial examples to machine learning models and datasets. Yuan *et al.* shows us that adversarial examples can be categorized along three dimensions. These dimensions are "Threat Model", "Perturbation", and "Benchmark" [1]. The threat model can be further broken down into Adversarial Falsification, Adversary's Knowledge, Adversarial Specificity, and Attack Frequency. Adversarial Falsification is the use of an adversarial attack to create either a false positive, where a human cannot recognize an image but the neural network classifies the image with high confidence score or a false negative, where an image is easily recognizable to a human but cannot be identified by the neural network [1]. In the case of malware detection, these two attack types correspond to a benign software sample being classified as malware or to malware being classified as benign software, respectively. The next

important threat model is Attack Specificity. This relates to whether the attack is targeted or non-targeted. In a targeted attack, the neural network is tricked using a specific, carefully constructed input in order to make the network misclassify the input. Oftentimes, the input is designed to trick the network into predicting the input into a specific class [1]. For example, if an adversary were attempting to bypass facial recognition, they would attempt to disguise the face as one that is an authorized user, tricking the system into allowing them access. On the other hand, we have non-targeted examples. With non-targeted examples, the adversary is attempting to find *any* input that will trick the neural network, rather than a specific input [1]. The aim may be to force the network to misclassify the input into an arbitrary class, thereby possibly allowing the input to slip past securities. This could potentially lead to disastrous consequences for neural network systems. Compared to targeted attacks, non-targeted adversarial examples are easier to implement due to the fact that there are more options to redirect the output to [1].

One of the final most important models of adversarial threats is Attack Frequency. Pertaining to attack frequency, we have One Time Attacks as well as Iterative Attacks. With One Time Attacks, only one iteration is needed through the neural network in order to optimize the adversarial example. On the other hand, Iterative Attacks must be run multiple times to update the adversarial example [1]. Iterative Attacks most oftentimes perform better than One Time Attacks, however, they require more interactions with the victim, as well as more computational time. Thus, for a task such as reinforcement learning, a computation-intensive task, choosing a one-time attack might be more feasible. It is also possible, however, that using an Iterative attack would tip off the victim network more than a one-time attack would.

A small amount of perturbation to a clean image is a fundamental requirement of adversarial examples. The attacker aims to create images with differences that are imperceptible to human eyes but causes performance degradation in machine learning and deep learning models. The two main perturbations are individual perturbation and universal perturbation. With individual perturbation, we generate a different perturbation for every clean input, which often takes up more time depending on the size of the dataset you are attempting to perturb. On the other hand, we have universal perturbation. With universal perturbation, the attacker creates one single perturbation that is applied to all clean input data across an entire dataset [1]. While many current attacks tend to generate adversarial examples individually, using universal perturbation make deploying an adversarial example in the real world much simpler. Thus, attackers do not need to change the form of their perturbation of the dataset once the input sample changes [1].

3. Related Work

With advancements in technology and a resurgence in artificial intelligence and machine learning, an arms race has been started between those who wish to attack and those who wish to defend neural networks. One of the most recent and advanced examples of AI-powered malware is called “DeepLocker” and was created by IBM Research [11]. The malware is embedded in a non-malicious carrier application where it would be downloaded by many different users. The malware then waits to unlock its payload until a specific user is recognized using a combination of facial recognition, voice recognition, and geolocation [11]. By utilizing a complicated deep neural network (DNN), the malware disguises and hides its trigger condition, payload, target class, and target instance [11]. Therefore, due to the highly complex nature of the neural network, even

if the malware had been found, it would be very difficult for malware analysts to determine who or what the malware was searching for. And without knowing these things, it would be impossible to decipher the trigger condition, meaning that the payload would never be unlocked and remain unable to be studied. In recent years, there has been an influx of research into artificial intelligence and neural networks. Much work has been done in the space of adversarial examples, proposing and classifying various types of attacks such as black-attacks, in which the attacker does not know anything about the structure of the network [12] as well as white-box attacks in which the attacker has some knowledge about the structure of the neural network. It has also been found that adversarial examples are transferrable among almost any artificial intelligence model. Shukla *et al.* first found that adversarial examples were transferable between the same neural network being trained on different datasets [12]. Meanwhile, [15] found that an adversarial example that effectively fools one model will most likely fool a completely separate model in the same manner or fashion. This is critical for black-box attacks as an attacker can use a substitute model to generate adversarial examples and then transfer them to the model in which they are invoking the attack. Yuan *et al.* explored possible defenses and countermeasures that could be employed against adversarial examples. According to their article, Yuan *et al.* claims there are two types of countermeasures against adversarial examples. The first is reactive, which involves detecting adversarial examples after deep neural networks are built, and the second type is proactive, which is the idea of making neural networks stronger and more robust so that adversarial examples cannot be generated for the model at all [1]. They go on to discuss three types of reactive countermeasures. The first is Adversarial Detecting. To do this, some researchers built a small, neural network as an

auxiliary to the original network that uses binary classification to predict the probability that an input to the original network is adversarial [1]. The second method is the Input Reconstructing. This method utilized Adversarial Detection to reconstruct adversarial inputs into clean data that will not affect the output of the neural network [1]. This is done through a denoising method, with an auto-encoder method to encode adversarial inputs to original ones, or even adding Gaussian noise to adversarial inputs to reduce their effect on the network [1]. Yuan *et al.* then discusses three proactive countermeasures to adversarial examples, the first of which is Network Distillation, in which neural network size is reduced by transferring knowledge from a large network to a smaller one to reduce the effectiveness of adversarial attacks [1]. The second proactive method is Adversarial Training, where the neural network is trained on adversarial examples in order to make the network more robust [1]. Finally, the third method of proactive countermeasures is "Classifier Robustifying.". In this method, Bayesian classifiers and Gaussian Processes were used to propose new neural networks called Gaussian Process Hybrid Deep Neural Networks (GPDNNs) [1]. These networks were found to be comparable to general deep neural networks (DNNs) in performance, yet more robust against adversarial examples [1]. Neural network robustness can be evaluated in a few different ways. One such way is to attempt to prove a lower bound, or generate attacks that can demonstrate an upper bound [18]. The former method, while effective, is much more complex to implement in the physical world. With the latter, if the attacks used on the neural network are not sufficiently strong enough, an upper bound may not prove useful [18]. Another powerful method to increase robustness of a neural network is Defensive Distillation. Defensive distillation is motivated by the goal to reduce the size of either Deep Neural Network (DNN)

architectures or ensembles of DNN architectures [20]. Doing this allows for the reduction of computing resource needs and in turn allows for deployment on resource constrained devices such as a smartphone [20]. This technique is done by extracting the class probability vectors that are produced by a DNN and transfer them to a second DNN of reduced dimensionality during training without observing a loss of accuracy [20]. On the other hand, Carlini and Wagner claimed that defensive distillation was no more resistant to targeted misclassification adversarial examples than an unprotected neural network [21]. They then showed that with slight changes to the attack model, they could achieve successful targeted misclassification on 96.4 percent of images by changing on average of 4.7 percent of pixels [21].

4. Methods

4.1. Object Detection

Our method of recognizing icons was to take an input image and parse through smaller crops of the image in order to make it easier to check the sections for web browser icons like Google Chrome, Microsoft Edge, Mozilla Firefox, and Opera. The classifier, starting at the top left of the input image and moving down and across, crops and sections the input image into overlapping 300x300 pixel boxes. In each of these boxes, the classifier searches for web browser icons and upon finding them, surrounds them in a small box and assigns a confidence level of recognition to the icon. Our classifier was trained to be able to recognize web browser icons both at any place on the desktop as well as any icons that have been pinned to the application tray. At the end of the program, the section in which each icon was found and recognized is put back on the screen so that the user can see them all at once. This project was made using an image classifier [16].

4.2. Creating Adversarial Examples using the Fast Gradient Sign Method and Image Processing

There are a large number of ways in which an adversarial example can be created. Introduced by Xie *et al.* in 2017, one such algorithm, named Dense Adversary Generation, or DAG, involved using a given image and recognition targets to create a perturbation in the image to confuse as many of the targets as possible [8]. As an overview, the authors implemented a random permutation function for each image independently, then the adversarial class label was set to a random permutation of the actual class label. They then minimized their loss function by making every target incorrectly predicted [8]. Their final step was optimization using a gradient descent approach. Other times, instead of designing and creating their own methods of building an adversarial example, researchers turn to common image processing techniques. Li *et al.* attempted to implement adversarial examples using techniques such as Gaussian Noise, Grayscale Image, Image Binarization, Salt-and-Pepper Noise, and Brightness Control [4]. The above-mentioned image processing techniques can be implemented using various Python libraries such as skimage, OpenCV, and PIL.

To create adversarial examples in our own project, we added noise to create our perturbed images using established image processing techniques as well as implementing the Fast Gradient Sign Method as proposed by Goodfellow *et al* [5]. Here, perturbed icons represent those icons that we have changed in an effort to fool the neural network, whereas unperturbed means the icon is original and remains unchanged. We worked with four types of image processing techniques in order to build our adversarial examples. The images we used as adversarial examples were screen captures of our desktop, with a perturbed image of a web browser icons pasted over top of the

background image. The type of additive noise we added to our images included Gaussian, Salt-and-Pepper, Poisson, and Speckle.

The Fast Gradient Sign Method (FGSM) is one of the most widely used and most effective methods of fooling an image classifier. It was first proposed by Goodfellow *et al.* in their paper "Explaining and Harnessing Adversarial Examples" [5]. The method is a linear way of perturbing inputs to an image classification model. Neural networks are often designed in linear ways to make them easier to optimize [5]. This suggests that neural networks are too linear to resist a linear form of perturbation. FGSM works by method uses the gradients of the loss with respect to the input image to create a new image that maximizes the loss. To accomplish this, FGSM finds how much each pixel contributes to the loss and then adds perturbation to that pixel accordingly. The equation for the Fast Gradient Sign Method of perturbation is given by $\eta = \in sign(\nabla xJ(\theta, x, y))$. In this equation, $\in$ is the smallest bit of an 8-bit image encoding once converted to real numbers, θ is the parameters of a model, x the input to the model, y the targets associated with x (for machine learning tasks that have targets), and J(θ, x, y) is the cost used to train the neural network. In our testing, we created our adversarial examples using FGSM in two sub-methods. The first, we ran the image through the FGSM model, which increased the size of the icon image to 300 by 300 pixels. The model then performed the perturbation on the icon and we then resized the adversarial image back to its original 48x48 pixel size. The second method, we padded the image to make it 300x300 so that the icon itself would not be enlarged. After creation of the adversarial image, we

then cropped the icon back out of the padding to get our final adversarial example to run through our image classifier. Original source code for the FGSM used in this project can be found at [17].

4.3 Gaussian and Salt-and-Pepper Noise

The first type of noise we worked with was Gaussian noise. Gaussian noise is a statistical noise that has a probability density function that is equal to normal distribution. This normal distribution gives us the Bell Curve as shown in Figure 1. This function can be written as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{(-\frac{(x-\mu)^2}{2\sigma^2})}$$

where σ and μ are representative of the mean and variance, respectively [4]. After Gaussian noise is added, the original image is still easily recognizable to a human. Since Gaussian noise is a form of additive noise, in which each pixel in the noisy image is the sum of the true value of each pixel and a random, Gaussian distributed value. The noisy image is described by $A(x,y) = H(x,y) + B(x,y)$ where A(x, y) is the function of the noisy image, H(x, y) is the function of image noise, and B(x, y) is the function of the original image [2]. The Gaussian noise we added to the chrome icon had a mean value of 0 and a sigma value of 20.0. These values provide us with a Gaussian (normal) distribution as follows,

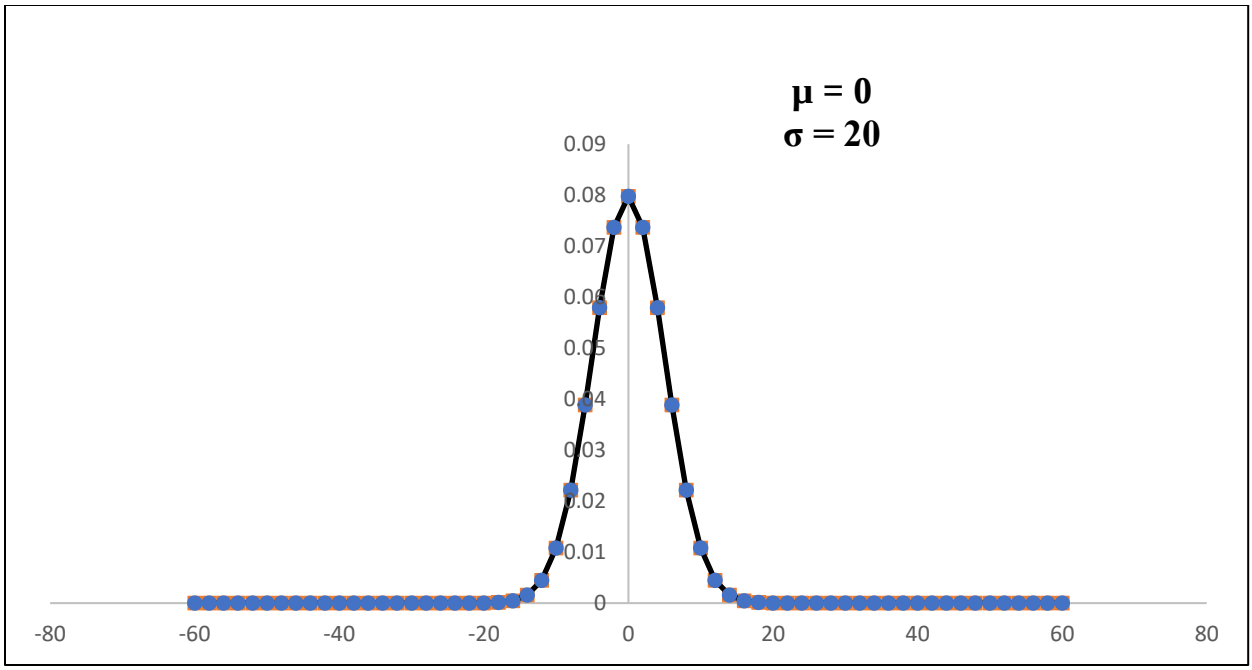


Figure 1. A Normal Distribution with a mean of 0 and standard deviation of 20

In addition to this, we also tested a slightly lower standard deviation value of 15.

Another form of noise we tested in our research is Salt-and-Pepper noise. Salt-and-Pepper noise is also commonly known as impulse valued noise or data drop noise due to the statistical drop of original pixel values [11]. This results in sharp and sudden disturbances in the image. To add Salt-and-Pepper noise to an image, we calculate the noise for a given pixel *(i, j, k)* as *Noise (i, j, k; p)*, where *(i, j, k)* represents the original, unperturbed pixel value and *p* represents the noise density value [4]. With Salt-and-Pepper noise, we have progressively dark pixels present in bright regions of the picture and vice versa [11]. In addition, not all pixels in the image will be perturbed as a result of Salt-and-Pepper noise. In the end, the image, as the name implies, has been sprinkled with a fine grainy layer of white pixels, the salt, and black pixels, the pepper.

4.4. Poisson and Speckle Noise

Our third chosen technique of noise addition was Poisson, also called Photon noise. Appearance of this noise is often the result of the statistical nature of many electromagnetic waves such as x-rays, gamma rays, and visible light [11]. Sources such as this have random fluctuations of photons resulting in an image with spatial and temporal randomness [11]. Poisson noise must always follow the Poisson distribution and can be modeled by $P(\lambda, k) = \frac{\lambda^k * e^{-\lambda}}{k!}$.

Finally, the final image technique we worked with was the addition of Speckle noise, also known as multiplicative noise. The probability density function of Speckle noise follows a gamma distribution and this type of noise can exist in an image similar to Gaussian noise [11]. The probability density function is given by $F(g) = \frac{g^{\alpha-1}e^{\frac{-g}{a}}}{(\alpha-1)!a^{\alpha}}$ [10].

5. Results

5.1. Unperturbed and FGSM Results

With an unperturbed image, the neural network was able to recognize Google Chrome, Microsoft Edge, Mozilla Firefox, and Opera icons on the desktop consistently with a confidence level of 99 percent. In the application tray, however, the confidence level for each icon suffered a minor loss, usually by approximately 2-4 percentage points.

The Fast Gradient Sign Method of creating adversarial examples was very effective in preventing the neural network from recognizing our web browser icons. In our tests using Google Chrome, with the resize sub-method, the image classifier was unable to recognize the icon. In the case of the pad and crop sub-method, the classifier was able to recognize the chrome icon with 77

percent confidence. Upon testing our methods with Microsoft Edge, with the pad and crop sub-method, we had a result of 67 percent confidence in recognition. Using the resize sub-method resulted in our image classifier being unable to recognize the Edge icon. The next icon we tested the FGSM on was Mozilla Firefox. Testing the resize sub-method on Firefox gave us a result of the network being unable to recognize the icon. In the case of the pad and crop method, our results were approximately 75-80 percent recognition. Finally, in our testing of the Opera icon, were found approximately 65 percent recognition confidence while using the pad and crop method. For the resize method, we found that the Opera icon was recognized sometimes with a confidence of around 75-80 percent, and in rare cases, it was recognized as Microsoft Edge with around 77 percent confidence. As a result of our testing using the Fast Gradient Sign Method, we were in most cases able to drastically lower the ability of our neural network classifier to identify web browser icons. In each of these tests, once the perturbed icon was pasted into the original input image, the icon on the background image was still very easily recognizable to human eyes. This perfectly fits one of the main criteria of our experiment.

5.2. Gaussian and Salt-and-Pepper Results

After adding the various types of noise, we were able to lower the classifier's confidence level in recognition of the web browser icons by differing amounts. The amount of drop-in confidence level depended on how much noise we wanted to add to the image. The first type of noise distribution that we tested on our image classifier was the Gaussian or normal distribution. For chrome, we needed a bit more noise than the other icons to drop the confidence of recognition by the same levels. After the addition of Gaussian perturbed chrome icon to our background, we

then ran the image through the classifier. Using a distribution with a mean of 0 and a standard deviation of 20, the image was still easily recognizable by a human, but the perturbed icon was recognized by the image classifier with a confidence level of around 81-85 percent. The level of noise in the image can be easily adjusted by changing the standard deviation. We also tested the perturbed image with a standard deviation of about 25. The image at this distribution had much more noise in it, and the image classifier was unable to recognize the icon in tests. For Microsoft Edge, a standard deviation of 15 was enough to bring confidence level of recognition to 82%, followed by the icon being unrecognizable at a standard deviation of 20. Next, for Mozilla Firefox, using a standard deviation of 15 dropped confidence level of recognition to 86%, and a standard deviation of 20 bringing it further down to 75%. Finally, in the case of the Opera icon, there were some unique results. The standard deviation of noise required to lower the confidence level of recognition to 75% was only 3. At a standard deviation of 4, the neural network was unable to classify the perturbed opera icon at all. Then, by increasing the standard deviation to 5, the network actually misclassified the perturbed Opera icon as a Firefox icon with a confidence level of 76%. Thus, from these results, I believe that using Gaussian distributed noise could prove an effective method of defense from AI.

Applying salt-and-pepper noise to our image was not quite as successful in blocking the neural network from identifying Google Chrome icons within the image. During our testing of the object detection network using an image generated with salt-and-pepper noise, the neural network was able to distinguish the icon for Google Chrome with an 88-90 percent confidence level. For Microsoft Edge, the confidence level of recognition hovered at around 95-97%. Mozilla Firefox

required a slightly larger amount of noise than other icons to drop the confidence level to 80%. Due to this fact, at a perturbation level that forced a recognition level of 80%, the Firefox icon was much more varied from the unperturbed image than in the cases of other icons. As for the Opera icon, the addition of salt-and-pepper noise was completely ineffective in reducing the confidence level of recognition by the neural network. When adjusting the amount of noise in the image, the Salt-and-Pepper method of noise addition was quite variable. A small edit to the noise function could drastically change the image. If the level was chosen such that the icon was still recognizable, the confidence of recognition hovered between 80 and 95 percent. With more change to the noise function, the icons were almost unrecognizable to the human eye and because of this, the image classifier was also unable to recognize the icons. This, however, is unhelpful in our research as we need the image to stay recognizable to human eyes. The introduction of salt-and-pepper noise to an image seems to be only slightly effective as an adversarial example for use in defense of a GUI based AI attack.

5.3. Poisson and Speckle Results

Poisson distributed noise was one of the most effective methods of tricking the neural network into failing to recognize the Google Chrome icon that we utilized. In addition to Salt-and-Pepper noise, it was also one of the most variable. Slight changes to the value of lambda in the Poisson distribution function caused large changes in image perturbation. To keep the chrome icon recognizable to humans, we used a lambda value of 500. Running this image through the classifier, the chrome icon was unrecognizable to the neural network. Using a much smaller lambda value of 100, the image was recognized by the neural net with the confidence of 88 percent, however, the

pixel colors were drastically reversed. For Microsoft Edge and with a lambda of 500, the network was unable to recognize the Edge icon. At a lambda value of 100, the Edge icon had very little noise, but the color had been reversed from blue to green. This brought the confidence level of recognition to 96%. In testing the Firefox icon, at a lambda value of 500, the Firefox icon was, like the others, unrecognizable. However, this was due to quite a large amount of resultant noise being added to the icon. With a lambda of 100, we reduced the confidence level to 92 percent. Lastly, in the case of the Opera icon, at a lambda value of 500, the icon was unrecognizable to the classifier. Unfortunately, however, this was a result of a large amount of noise added to the icon by the Poisson function. At a lambda value of 100, as with the other icons, the colors were reversed. Thus, the image classifier actually recognized the icon as Google Chrome with 74 percent confidence. While Poisson additive noise was effective in preventing the classifier from recognizing our icons or even misclassifying them as other icons, the resulting perturbed image, once run through the classifier, had either drastically reversed colors or had a large amount of visible noise. These may not be what we want while testing our theory. At this point in time, testing shows that Poisson distributed noise is an effective method of fooling our image classifier and has the potential to be effective in defending from AI malware.

The final method of noise addition we employed was speckle noise. After testing, speckle noise was one of the methods that allowed for the least amount of our control over the amount of noise in the icon. After creating and adding the speckle noise to the Google Chrome icon and pasting it back into our background image, the icon was almost completely overpowered by the noise. The shape of the icon was still the same, but no was not recognizable as chrome to the eye

in any way. This directly led to the image classifier being unable to recognize the icon as well. Upon adding speckle noise to Microsoft Edge, the icon had a lot of noise in it but was still recognizable as Edge. This reduced confidence in recognition by the classifier to 70 percent. With Mozilla Firefox, as with Google Chrome, the icon was overpowered by the speckle noise, making it unrecognizable to both human eyes and the classifier. Finally, when testing Opera, the icon was still mostly recognizable as the Opera icon and the network was unable to classify the icon. In two out of the four icons we tested, speckle noise completely overpowered the icon. These two were Google Chrome and Mozilla Firefox. It appears that due to the whitespace in Microsoft Edge and in Opera, these icons were still recognizable to the human eye after the addition of speckle noise. However, since we want our adversarial icons to still be easily recognized by the human eye for all cases, speckle noise is not a viable option in our experiment to fend off a GUI-Attack from an artificial intelligence image classifier.

Table 1: Sample Images for Perturbation Types

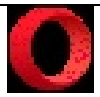
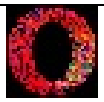
Unperturbed	FGSM (Resize)	FGSM (Pad and Crop)	Gaussian std = standard dev.	S&P	Poisson	Speckle
			 std= 20  std = 25		 λ = 500  λ = 100	
			 std = 15  std = 20		 λ = 500  λ = 100	
			 std = 15  std = 20		 λ = 500  λ = 100	
			 std = 3  std = 4		 λ = 500  λ = 100	

Table 2: Summary of Results

% Recognition indicates how well the model accurately classified the icon

	Unperturbed	FGSM (Resize)	FGSM (Pad and Crop)	Gaussian std = standard dev.	S&P	Poisson ***	Speckle *****
	97-99%	0%	77%	81-85% std= 20 0% - std = 25	88-90%	0% - λ = 500 88% - λ = 100	0%
	97-99%	0%	67%	82% - std = 15 0% - std = 20	95-97%	0% - λ = 500 96% - λ = 100	70%
	97-99%	0%	75-80%	86% - std = 15 75% - std = 20	80% **	0% - λ = 500 92% - λ = 100	0%
	97-99%	75-80% *	65%	75% - std = 3 0% - std = 4	99% Ineffective	0% - λ = 500 74% - λ = 100 ****	0%

* In rare cases, while using this method on Opera, the icon was recognized as Microsoft Edge with around 77% confidence

** Firefox required more noise to drop recognition level, icon became much less recognizable from original image

*** At λ = 100, icon colors were often inverted once put through the image classifier

**** For Opera at λ = 100, icon was recognized as Google Chrome with 74% confidence

***** Icons were mostly overpowered by speckle noise, making the method ineffective for our theory

6. Conclusion

Despite their complicated and advanced nature, neural networks for image classification have the potential to be readily fooled by properly constructed adversarial examples. These adversarial examples can prevent the image classifier from identifying the input image at all or even force it to recognize it as something else with a high level of confidence. In the future, with advancements in technology, we will begin to see malware and viruses that utilize artificial intelligence and image classification in order to recognize specific icons and open them to steal private data. A proposal for future research may be the construction and possible testing of such a computer malware that performs the functions stated above. In doing this, we could learn more about how it would work as well as how we would be able to effectively and efficiently prevent and defend against it. One problem with this however, may be that releasing research on this topic may provide people with the knowledge and means to create their own AI based malware, more easily opening the door for people to implement one on victim systems in the real world. This happened with the DDoS attack on Dyn in October 2016, in which a hacker used the code for the "Mirai Worm" to attack and take down a large number of major websites worldwide. In our research, we proposed that we could use adversarial examples in an attempt to hide and mask our web browser icons and prevent a neural network from recognizing them while keeping them as close to the original icon as possible, thus protecting our information. In our testing, we found that out of the image processing techniques we used, Gaussian and Poisson were the two most effective. However, it must be noted that with Poisson distributed noise, it was common for icon colors to be reversed in the process of adding the noise to the image. The Fast Gradient Sign Method was the most effective in our testing, with the perturbed icon still looking very similar to

the original and easily recognized by human eyes after being pasted back into the desktop background. We believe our results have shown that adversarial examples have great potential as a tool to be utilized in cybersecurity in the very near future.

References

1. X. Yuan, P. He, Q. Zhu and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805-2824, Sept. 2019.
   doi: 10.1109/TNNLS.2018.2886017
2. Swain, A. (2018, August 28). "Noise in Digital Image Processing". Retrieved from *https://medium.com/image-vision/noise-in-digital-image-processing-55357c9fab71*
3. Geng, D., & Veerapaneni, R. (2018, January 10). "Tricking Neural Networks: Create your own Adversarial Examples". Retrieved from *https://medium.com/@ml.at.berkeley/tricking-neural-networks-create-your-own-adversarial-examples-a61eb7620fd8*
4. Li, X., Ji, S., Han, M., Ji, J., Ren, Z., Liu, Y., & Wu, C. (2019). "Adversarial Examples versus Cloud-based Detectors: A Black-box Empirical Study". *arXiv preprint arXiv:1901.01223*.
5. Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". *International Conference on Learning Representations (ICLR), 2015*.
6. Florian Tramer, Nicolas Papernot, Ian Goodfellow, and Patrick McDaniel Dan Boneh. "Ensemble Adversarial Training: Attacks and Defenses". *ArXiv preprint arXiv:1705.07204, 2018.*
7. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille. "Adversarial examples for semantic segmentation and object detection". *arXiv preprint arXiv:1703.08603, 2017*.
8. Grosse K., Papernot N., Manoharan P., Backes M., McDaniel P. (2017) "Adversarial Examples for Malware Detection". In: Foley S., Gollmann D., Snekkenes E. *(eds) Computer Security – ESORICS 2017. ESORICS 2017. Lecture Notes in Computer Science, vol 10493. Springer, Cham*
9. Kurakin, I. Goodfellow, and S. Bengio. "Adversarial examples in the physical world". *arXiv preprint arXiv:1607.02533, 2016.*
10. Boyat, A. K., & Joshi, B. K. (2015). "A review paper: noise models in digital image processing". *arXiv preprint arXiv:1505.03489*.

11. Stoecklin, M. P. (2018, August 13). “DeepLocker: How AI Can Power a Stealthy New Breed of Malware”. Retrieved October 6, 2019, from *https://securityintelligence.com/deeplocker-how-ai-can-power-a-stealthy-new-breed-of-malware/*.
12. Shukla, S. N., Sahu, A. K., Willmott, D., & Kolter, J. Z. (2019). “Black-box Adversarial Attacks with Bayesian Optimization”. *arXiv preprint arXiv:1909.13857*.
13. Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian J., and Fergus, Rob. “Intriguing properties of neural networks”. *CoRR*, abs/1312.6199, 2013.
14. Rauber, J., Brendel, W., & Bethge, M. (2017). “Foolbox: A python toolbox to benchmark the robustness of machine learning models”. *arXiv preprint arXiv:1707.04131*.
15. N. Papernot, P. D. McDaniel, and I. J. Goodfellow, “Transferability in machine learning: from phenomena to black-box attacks using adversarial samples,” *CoRR*, vol. abs/1605.07277, 2016.
16. Z. Tuttle, “TensorFlow-MachineLearning-Research”, *https://github.com/criogenesis/Tensorflow-MachineLearning-Research*
17. “A Step-by-Step Guide to Synthesizing Adversarial Examples”, *https://www.anishathalye.com/2017/07/25/synthesizing-adversarial-examples/*
18. N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," *2017 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, 2017, pp. 39-57. doi: 10.1109/SP.2017.49
19. Athalye, A., Engstrom, L., Ilyas, A., & Kwok, K. (2017). “Synthesizing Robust Adversarial Examples”. *ICML*.
20. Papernot, N., McDaniel, P., Wu, X., Jha, S., & Swami, A. (2016, May). “Distillation as a defense to adversarial perturbations against deep neural networks”. In *2016 IEEE Symposium on Security and Privacy (SP)* (pp. 582-597). IEEE.
21. Carlini, N., & Wagner, D. (2016). “Defensive distillation is not robust to adversarial examples”. *arXiv preprint arXiv:1607.04311*.