2 APR 2025

Getting Started with AI Hacking: Part 1

| Brian Fehrman

Brian Fehrman has been with Black Hills Information Security (BHIS) as a Security Researcher and Analyst since 2014, but his interest in security started when his family got their very first computer. Brian holds a BS in Computer Science, an MS in Mechanical Engineering, an MS in Computational Sciences and Robotics, and a PhD in Data Science and Engineering with a focus in Cyber Security. He also holds various industry certifications, such as Offensive Security Certified Professional (OSCP) and GIAC Exploit Researcher and Advanced Penetration Tester (GXPN). He enjoys being able to protect his customers from "the real bad people" and his favorite aspects of security include artificial intelligence, hardware hacking, and red teaming.

You may have read some of our previous blog posts on Artificial Intelligence (AI). We discussed things like **using PyRIT to help automate attacks**. We also covered the **dangers of RAG systems**. We may have gotten a little ahead of ourselves. What we didn't cover is the more fundamental aspect of, "What does this all mean and how do I get started?"

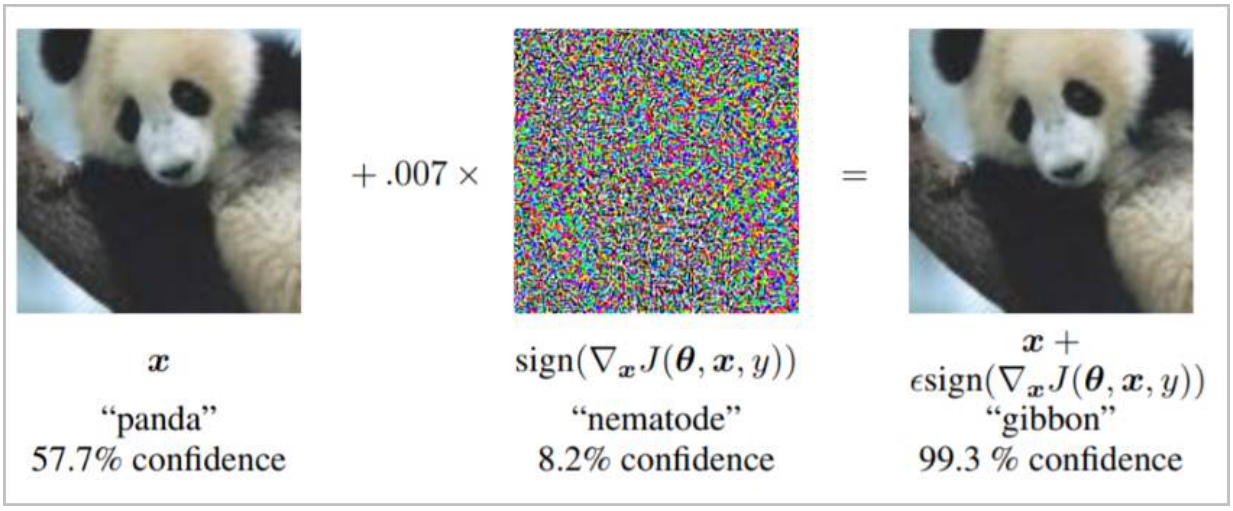
In this series, we will define AI "hacking" and the various types/methods of AI hacking. We will kick off the series with a more "traditional" form of AI hacking that focuses on classifier models. Future blogs will then pivot to LLM hacking. Those future blog posts will discuss prompt injection attacks, excessive agency, and sensitive data access. For now though, let's look at classifiers.

With that, let's hop in!

Image Classification Hacking

Let's take a common example that you may come across if you start researching AI hacking. Below, we have an image classifier that was trained to determine what animals are present in images. On the left is the original

image, which is correctly identified as a panda. The pixels are strategically manipulated by adding what appears to be noise to the image. The result is a new image that appears to humans to be a panda…but the classifier says that it's a gibbon.



Tricking Image Classifier with Noise Perturbations
(Source: https://pytorch.org/tutorials/beginner/fgsm_tutorial.html)

A Real Gibbon

So, what happened? Unlike humans, computers can only deal with literal bits of information. Sure, we can have a philosophical discussion there on how humans process information, but let's skip that for now and agree that people are able to look at the bigger picture. We can look at a scene and quickly identify what's in it, with relatively high accuracy, without inspecting every minute piece of information available. Computers can't do that. Computers piece together bits of information into groups, compare them to other bits of information that have been grouped, and try to determine what groups of bits are most similar. What happens, though, if we just slightly modify some of those bits to more closely match bits seen in other groups? We get the result above: an image that is misclassified by a computer, but a human views as nearly identical to the original image.

So now what… we've gotten a panda to be classified as a gibbon? Sure, that is just one example. What if instead of a panda, it's a stop sign? And the classifier controls a self-driving car? Well, this exact stop-sign misclassification scenario has been proven. Researchers were able to shine a specific light pattern onto a stop sign to get it to be classified as a 30 MPH speed sign. The implications of misclassifying a stop sign are self-evident. Researchers have also been able to perform the same attack by strategically placing stickers on a stop sign.

Perturbations on a sign, created by shining crafted light on it, distorts how it is interpreted in a machine learning system. (Source: https://arxiv.org/pdf/2108.06247.pdf)

Perturbations on a sign, created by shining crafted light on it, distorts how it is interpreted in a machine learning system. (Source: **https://arxiv.org/pdf/2108.06247.pdf**)

The prior attacks are known as a*dversarial examples*. Adversarial examples exploit the fact that AI systems are trained on limited sets of samples and must generalize future predictions based on that limited data.

Mitigating misclassifications of adversarial examples is an active area of research. One approach is to train the model on adversarial examples. Another approach is to prevent overfitting of data during training as it allows classifiers to better generalize to new data, which makes them less susceptible to these types of adversarial attacks.

Hacking Malware Classifiers

Ok, the image classification is neat, but what about something more related to computer systems? Ever hear of anti-virus (AV) products that use machine learning? Endpoint Detection and Response (EDR) that utilizes AI? Any guess as to how those might work? At their heart, AV systems that utilize AI are just classifiers. They've been given examples of what malware looks like. They've seen steps that are taken by threat actors. They are then asked to determine if what they are currently looking at appears to be similar to what they've seen before. The AI security products must decide if what they see looks like malicious or benign behavior. Sure, this is an oversimplification, but it's not far off.

What if we can give malware some characteristics of benign software but keep its malicious functionality? Let's take an example from a few years ago. Researchers found that they could easily bypass Cylance's AI powered AV product (**https://www.vice.com/en/article/researchers-easily-trick-cylances-ai-based-antivirus-into-thinking-malware-is-goodware/**). Was it through direct kernel calls? Clever encoding? Self-inflicted ROP attacks? No! They took strings from a benign program and appended them to their malware file. The Cylance classifier inspected the malware, saw that it had a lot of similarities to benign programs it had seen before, and made that call that the malware was likely benign.

Pushing the characteristics, or features, of data over a decision boundary is one key part of AI hacking. As we mentioned, classifiers can only really make decisions based upon information that they've seen before.

Let's look at another example (and a shameless plug of some of my prior work). My PhD work focused on pushing VBA malware over the decision boundary into benign territory (or, at least, "not detected" territory). The classifier that I focused on did analysis on both semantic (what code is doing)

and syntactic (how does the code look) code attributes. The idea of the work was to modify the structure and attributes of the code to more closely align with code samples that the classifier predicted as benign. Some of the features included identifier length, entropy of characters in the identifiers, uppercase/lowercase ratio of characters in identifiers, and number of lines in functions. Without going into the gory details, the attack worked like this:

- Determine common attributes of VBA files classified as benign
- Determine common attributes of VBA files classified as malicious
- Modify the code within malicious files to more closely match the attributes of files classified as benign

That is all much easier said than done. Modifying pixels in images is one thing, as in the end, you still have an image. Modifying code is a whole different animal as you need the code to still function after it is modified, otherwise the effort is basically worthless. Performing evasive attacks in code in an automated fashion is currently an open research problem that is gaining more attention. In this case, I was able to flip the classification of approximately 2,566 files out of approximately 3,800 from malicious to benign by shifting code attributes to more closely align with those seen in benign files.

Benign	Malicious	Percent Flipped from Malicious to Benign
2566	1313	66.15%

Results of Shifting Code Attributes of VBA Files

Clone-a-Classifier (Model Extraction Attacks)

Another attack is model cloning. Cloning a model can have a few purposes, such as reducing the cost of interacting with one. Anybody who has looked into purchasing API credits for some of the popular models can attest to the

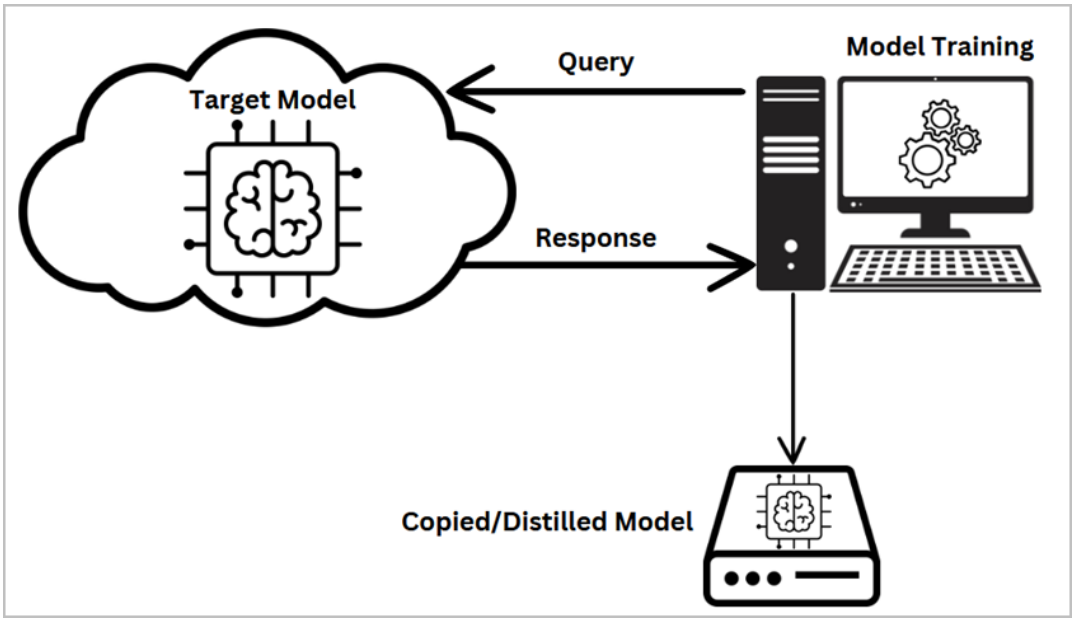
expense. If you have your own model that works similarly, you can avoid the long-term costs. In some cases, people might even try to monetize the use of their copycat model. Selling access to a cloned model is basically the AI equivalent of selling a knock-off Rolex. Just like with knock-off products, you will find varying degrees of quality in the copycat models. There is actually a high-profile case of alleged model copying going on right now with allegations that DeepSeek copied OpenAI's ChatGPT model. I am not arguing for or against those allegations, but those are great timing for this blog post for a real-world example that I can point to.

People might also copy models to avoid giving data to the target model and those who might be monitoring activity. Let's touch on the previous section of this blog post. If you are developing evasive malware, you likely don't want a model to see your final sample before you use it in your intended, real-life scenario. If you can make a copy of the model, then you can locally test against your model and refine your malware before deploying it against the live target model.

There are also some legitimate purposes for copying a model. You may run across the term *model distillation*. In essence, model distillation is creating a small model based on a large model. The goal is to have the smaller model perform similarly to the larger model on one or more tasks. Having the smaller model means you need fewer computing resources to gain faster performance. Think of it like creating a study guide from an entire textbook. The textbook contains a lot more information, but it also utilizes a lot more paper (or hard drive space, I guess in this day) and is more cumbersome to navigate when you need specific bits of information. A study guide captures the main highlights and essence of the data in the textbook but in a much smaller, easier to use footprint.

In any of the aforementioned cases, a *model extraction attack* is what allows people to copy models. So how does it work? The main flow is that you:

1. Send data to a target model
2. Get responses from the target model
3. Use the data and the responses from the target model to train your own model



Overview of Model Extraction Attack

Preventing model extraction attacks can be difficult as you need to let users interact with your model. One of the key defenses against extraction attacks is rate limiting.

Model Inversion (Training Data Extraction) Attacks

Interacting with a model usually means giving it data and getting a response that is based upon comparing your data with data that the model was trained on. In some cases, you can trick models into revealing the actual data that they were trained on.

How is this attack achieved? Consider an instance where we have a facial recognition system. The system takes an image and returns a confidence score on how closely it matches a known image. Without getting into the

heavy math details (e.g., gradient descent), here is a basic overview of an attack:

1. Submit an image with randomized pixels to the classifier
2. Observe the probability returned that it matches with a known image
3. Adjust the pixels in the image and resubmit
4. Observe the change in match probability
5. Use the changes in probability to guide future updates to image pixels to attempt to increase the probability score
6. Iterate through steps 3-5 until some exit condition (e.g., number of iterations, satisfactory match probability, etc.)

The result of this attack is that you may be able to reconstruct an image that closely matches an original image in the dataset that was used for training. This could allow for determining who has access to certain systems or data.

Left) Recovered Image; Right) Original Image
(Source: https://franziska-boenisch.de/posts/2020/12/model-inversion/)

Here are some additional examples of applications of model inversion attacks:

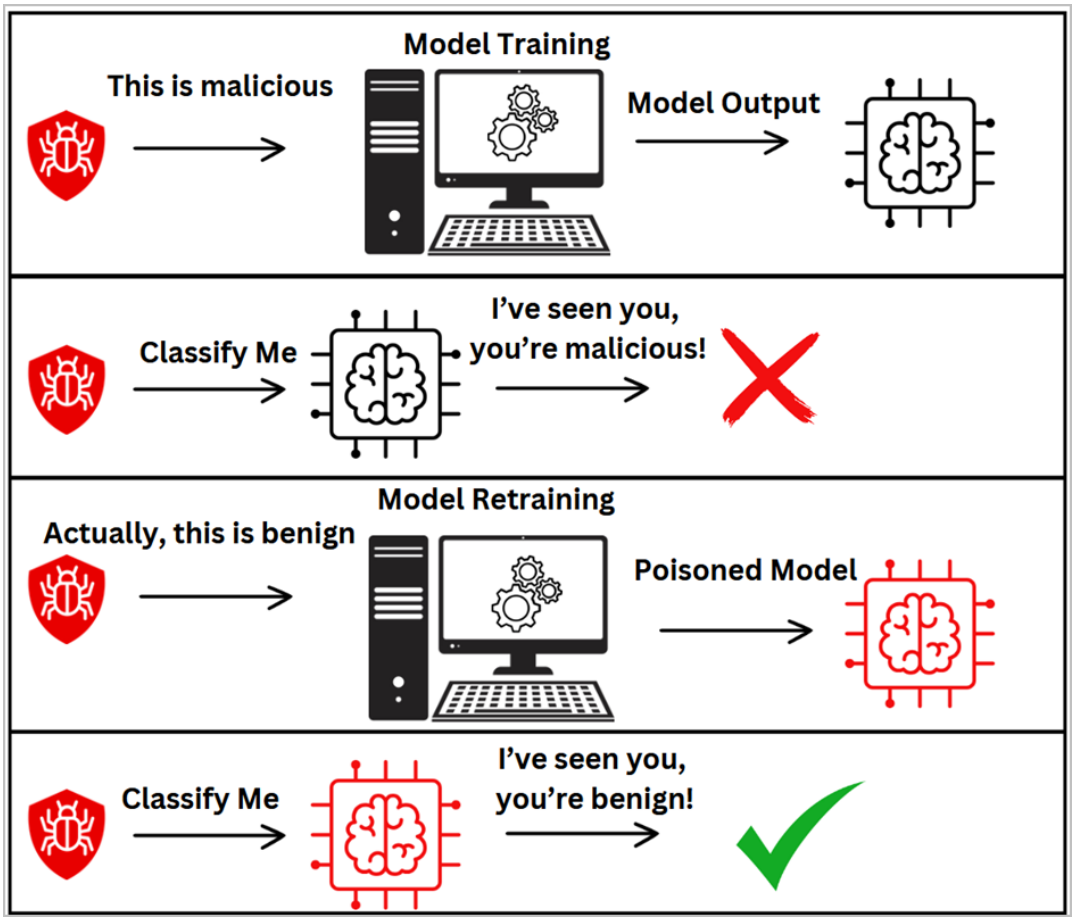
- Recovering sensitive medical data
- Extracting sensitive phrases or PII
- Recovering biometric signatures other than facial images, such as speech patterns

In addition to rate limiting, one of the main defenses against model inversion attacks is to use *Differential Privacy*. Differential privacy strategically adds noise to the training data in such a way that the original data cannot be recovered. Systems can also opt to return binary data (e.g., true/false) instead of a continuous range of probabilities.

Data Poisoning Attacks

Models are only as good as the data that they are trained on. What happens if you purposefully mess with the data that is being used for training? The result of tampering with training data can be a model that has a higher probability of misclassifying a given sample. Let's go back to the malware classification scenario.

There are a few ways to go about data poisoning attacks. Arguably, the most straight-forward way is if you can access the location of the training data to either inject your own samples or to modify labels of existing samples. In this case, we either add malware samples that we label as benign or flip the labels of existing malware samples from malicious to benign. When the model is retrained on this data, it will begin classifying known-malware as benign.



Examples of Data Poisoning Attack

You might think that gaining access to training data seems far stretched. But consider public sources that you can upload to, such as Hugging Face. Many models are constantly scraping data from the internet. Threat actors can potentially leverage this public scraping to insert purposefully mislabeled data.

Launching off the public data source avenue, some models also dynamically retrieve data when making decisions or providing responses to users. Dynamic retrieval is common in LLM systems that utilize Retrieval-Augmented Generation (RAG). I won't go into detail here as we already have a blog on that (**https://www.blackhillsinfosec.com/avoiding-dirty-rags/**). Understand though, the implications of threat actors gaming the search results to increase the odds of their malicious data being retrieved when responding to user queries.

Another vector for poisoning data is systems that incorporate user feedback

or user-supplied data into their retraining. As an attacker, you can slowly move the needle by purposefully giving incorrect feedback, or by slowly changing your data sample in the case of attacking malware classifiers.

Companies can defend against these attacks by vetting the data sources that they use. Storage of data internally should follow traditional security protocols in limiting and monitoring access, as well as ensuring systems are frequently updated to mitigate security vulnerabilities. Running outlier classifications on datasets can also potentially reveal anomalous samples that may need a closer look.

Conclusion

This blog post introduced the concept of AI hacking by discussing common attacks for various machine learning systems. The hope is for you to understand that machine learning is not perfect and some of the ways that threat actors might exploit those imperfections. This blog is the first in a series and focused more on attacking classifier systems. Our upcoming posts in this series will focus on the much hotter topic of attacking generative AI LLM systems. We will dive into how LLM systems really work, and how we can poke at them and leverage them for attacks. Stay tuned for the next blog in this series on getting started with AI hacking!

REFERENCES

- **https://www.blackhillsinfosec.com/avoiding-dirty-rags/**
- **https://www.blackhillsinfosec.com/using-pyrit-to-assess-large-language-models-llms/**
- **https://link.springer.com/chapter/10.1007/978-3-031-67447-1_19**
- **https://ieeexplore.ieee.org/document/10778829**

Go-Spoof: A Tool for Cyber Deception

Offline Memory Forensics With Volatility

890 Lazelle Street, Sturgis, SD 57785-1611 | 701-484-BHIS (2447)

LINKS

SEARCH THE SITE