SEI Webcast

Threats for Machine Learning

by Mark Sherman

Shane McGraw: Hello. Welcome to today's SEI webcast, Threats for Machine Learning. My name is Shane McGraw, outreach team lead here at the SEI, and I'd like to thank you for attending. We want to make our discussion today as interactive as possible, so we will address questions throughout today's talk. You can submit those questions in the YouTube chat area and we will get to as many as we can.

Our featured speaker today is Dr. Mark Sherman. Mark is the technical director of the Cybersecurity Foundations Group at the SEI. His team focuses on fundamental research on the lifecycle for building secure software and on data analytics in cybersecurity. Before coming to the SEI, he was at IBM and various startups, working on mobile systems, integrated hardware-software appliances, transaction processing, languages and compilers, virtualization, network protocols and databases. Mark received his undergraduate degree from MIT and his PhD in computer science from Carnegie Mellon University. Now I'd like to turn it over to Mark Sherman. Mark, good afternoon. All yours.

Mark Sherman: Great. Thank you very much, Shane, and good afternoon to everybody. Welcome to Threats to Machine Learning Applications. My team, the Cybersecurity Foundations team, is part of the larger CMU campus, which has extensive research in computer science, and in addition to what you see of the rankings of number one in computer science, AI, U.S. News has recently ranked us number one in cybersecurity as well. And within that larger community, I'm part of the Software Engineering Institute, a research center focused on software engineering and cybersecurity and artificial intelligence.

And within the Software Engineering Institute in particular, my team is part of the CERT division, and the CERT division, established way back in the '80s, responsible to (inaudible), has been focused on cybersecurity issues for decades, and as the technology and the applications of software have evolved, so has the attention of where we focus cybersecurity. With the increased use of machine learning and AI applications, the need to understand how to defend those applications and how to make them more robust has become a top priority. So what I'm going to share today is an overview of some of the issues that one might face in trying to build a more robust machine-learning-based system.

So to begin with, I will just give you some examples of what it looks like to try to fool these systems. Many people have seen these before, but if you haven't, they sometimes cause some amusement and therefore I thought I'd start by just sharing those with you.

For example, one is a stop sign that fools pattern recognition systems into thinking that it's a speed limit sign, just by adding those little black and white squares.

Another common example, the ability to make what looks like a turtle to always appear like a rifle no matter you are looking at it because, again, the machine learning system has been compromised.

Another example, based on exfiltration that is looking at the model and understanding how it works is that you can make the system think that anybody is Milla Jovovich by just wearing those magic glasses, and in fact by developing different airs of glasses you can look like anyone that the system can recognize.

Another example that's used a lot is how to hide from facial recognition systems. In this case, it's just by putting on this special tee shirt that it fools the system into thinking that you're actually not there; there's no face to be recognized.

And perhaps the granddaddy of all these kinds of examples is the panda, when sprinkled with magic noise, makes the system think that it's a gibbon. Now, we're not going to go into a lot of the deep statistics that either of these exploits relied on, or how exactly you defend against them. My purpose today is to give you a basic understanding of how these systems are built, and in an abstract way, how they can be attacked and what an attack looks like. The goal is to provide enough knowledge that you can have a good conversation with the data scientist or machine learning expert in order to understand the tradeoffs that might be made, the risks that you might be taking as you deploy and build these systems. There's nothing that's foolproof, and so many times you'll be faced with tradeoffs, and by understanding a little bit about what's going on under covers, you can have a good conversation about where we'd like to make that kind of (inaudible) choice.

So to do that, we should first understand what the machine learning attack surface is; that is, what are places that an adversary might go in trying to attack all these machine learning applications. So we'll start with a very abstract view of the machine learning application, and this comes from the book that's mentioned below.

As I said, I'm not going to be going into the math or the statistics behind all these kinds of things, but this book does contain a very good discussion of a lot of the math behind what we'll be talking about today. So if you really want to get into the details, you can do so. And we'll step through this whole development process and departmental process for machine learning applications and take a look at how an adversary might view these as places that they might attack.

So let's start at the very beginning, what's called the subset of domain data, and just for sake of discussion, let's say we're trying to build a system that can distinguish between bananas and oranges-- not exactly a practical application but something that we can easily talk about. So the first thing is you're not going to have pictures of all apples and all oranges that ever existed.

You'd have to pick some particular subset of them in order to look at them. If we had the ability to control what that subset was, you could, for example, exclude particular things that you don't want recognized that might be subversive. You want to somehow get bananas with black spots in but you don't want them to be recognized as such, and so you might exclude that kind of data.

Historically-- it's not been a cyberattack but there have been issues with subsets of data in very early language systems-- I'm sorry, visual recognition systems, used to use basically white college graduates based in Silicon Valley who were software engineers as the collection of faces to look at. Obviously that subset-biased those systems tremendously. We know better in some cases, but it's still a place that an adversary can attack.

Just to take a slight detour here, for the most part I'm going to be talking about a particular subset of these systems, something called supervised learning systems. That's just for simplicity of discussion, but in this particular case it's worth mentioning there's another kind of system called active learning systems. The picture shown here shows a very static view of the world. I take my bananas and oranges and then I do some processing at the end; I wind up with a program that can distinguish them. Active learning systems don't start with only a static collection of domain data but rather they allow new data to be added in as the system is running in order to continuously train it. As you see, you can start to feed in bad data into that domain and it'll start learning very bad things. There's other talks we have that goes into that a little bit more, but one common example that was talked about in the press was a bot that Microsoft put together that basically learned how to be very obscene based on this kind of drift as the domain data got polluted.

The next part of building an application is basically called feature selection, which is based on measurement. Again, let's take a look at our bananas and oranges. Looking at those bananas and oranges, we can take certain kinds of measurements that we're actually going to use for the system. Let's say, for example, we've got weights, we've got color, we've got size, just to pick some examples. From that, we can then develop some features, which are usually some combination or selection of the measure-- and again, just for sake of argument, we can say that based on that data, based on those measurements, we're going to take color and we're going to take density; rather than both size and weight we'll just convert that into the two things. If you can control what features are important-- again, take the bananas. They somehow don't allow color in, so I don't recognize black spots, then I can have the system not behave correctly when it's these bananas with black spots in it. So this is another place that an adversary can go and try to alter how the system can behave.

The next part is in the training data, and we'll talk a little bit about this more later, but once you have your collection of features based on the subset of the data you have, you split it into two piles. One pile you're going to use to train the system so it learns how to recognize, again, oranges and bananas; and the other side, as we'll see in a moment, is how you're going to be used to test the system to see whether it adds effectively words, how to distinguish between oranges and bananas. If the training data you pick is selected so that it only looks at certain kinds of data, only looks at perfectly round oranges, or shrunken oranges with a very high density in them, then you will bias how the system makes selections; or, if it's adversarial, not so much but actually cause it to make the wrong decision.

Similarly, on the test side, you can try to feed a system change data that will make the testing look like a system that is behaving correctly behave incorrectly, and you can do this in two ways. The testing data-- and again, we're talking about supervised learning just for simplicity. We can generalize this to other kinds as well, but for supervised learning we've actually got tags associated with each of the domain data. So I have a picture of an orange and I can know that it's an orange; I have a picture of a banana and know it's a banana. That's (inaudible) truth that's on the bottom there. You might have an adversary that attacks it. You can label things that are bananas as oranges, and as a result have it miscalculate what it thinks is an orange and a banana accordingly.

Another place to attack is in the model-building algorithm and the built model. We combine these together. In other words, the system that you use that takes all this data and then develops the resulting model itself can either be statistically attacked-- that is, it purposely does things that will cause it to misbehave-- I'll take, again, a very simple kind of example. You might want to build a system, you have control over the algorithm, to say that anytime it sees a white dot in the upper left-hand corner, it's always going to call that an orange, and so that provides this backdoor for an adversary to actually force the system to always call something an orange by just making sure it puts a white dot in that right place.

More conventionally, this has already appeared as a problem in ordinary software systems-- think about a compiler, for example-- where the program itself is fine but the compiler, in addition to compiling the program, adds a piece of malware to the final output. So to the model-building algorithm you could just add a piece of malware into the final model, having nothing to do with machine learning, but nevertheless make the system susceptible to attack.

And finally, there's the loss calculated. The way that you understand whether a model has been adequately trained or not is you take a look at the result of the training data-- that is, what it thought was bananas and oranges. You then take a look at the testing, for which you have the truth-- right? That's part of the testing data-- and you take a look and say, "How close was the testing data that we fed in?" I fed in all these oranges. Did it label them all oranges? I fed in all this testing data of bananas. Did it label them all as bananas? If so, then you have a very small error and you say that's a very well-trained model. If it doesn't match, those are the models not well-trained and we have to go back and figure out a better way to train the model with different algorithms, different data, different features, and so on. But if you can control that error, you might feed in what looks like everything that's oranges or everything that's bananas and you'll declare the error to always be zero, and therefore you'll have a system that misbehaves, because you're able to control the error analysis, how well you think the system is working.

Now, there's a particular kind of machine learning system that has additional risks associated with it, and that is a subset which we'll call transfer learning systems, and we like to go into this because it's a very popular way to build these machine learning applications that's a little more directed than this general notion of the entire lifecycle of machine learning application development.

So here's an example, a little blown up, of a machine learning application-- it's sometimes called a deep neural network structure-- and the way the systems work is they start with data-- so in the example we gave, for example, of the far left, there would be two boxes there, two circles there. One would be density; one would be color-- just, again, as an example-- and then there's some hidden layers which try to abstract different features between them, so how high does the density go, how rich is the color, and so on. In this example with faces, you can see a little more about what's going on. There's some edge detection in the first layer; there's some feature detection in the second one; and then after you start getting closer to the people. At the very end you have the output layer which is actually defining what you're looking for. So again, in our example, you might have one of those circles that says, "I found an orange," and another circle that says, "I found a banana."

Now, what's going on inside of those layers is actually a calculation, and we're not going to go into how all these numbers are calculated, but the point is that as you feed examples into the left side, the statistics on the inside change those numbers that are in the nodes and change the weights of those edges as those numbers propagate through the system from left to right, eventually getting to the right-hand side that gives, in this case, a probability of whether you're in the top circle or the bottom circle. As you do those kinds of calculations it gets better and better, and then you use the error function that we talked about before to decide whether you've got it trained well enough or not.

Now, this is a fairly simple example. Other examples are far more complicated. To give you an idea for how big this can be, in the recently released GPT-3 neural network, that has 175 billion parameters. That's billion with a B. So you see on this chart maybe a dozen or so parameters. Imagine if you had to do that calculation for billions and billions of those parameters. That is a lot of work, but that is why transfer learning has become so interesting.

So transfer learning starts by having a trained model. So here, again, abstract of the model out. It starts with an input, has some output; it has all those hidden layers in between, and because it's been trained, all those parameters, the hundreds of millions of parameters, have already been set. So, for example, we've built one of these systems for oranges and for bananas and we have all those numbers calculated.

The way transfer learning works is you take that same system and you throw away that final stage, the stage that says, "Ah, I've recognized an orange," or "I've recognized a banana," because now we want it to choose between oranges and apples, again, hypothetically.

So what we'll do is we'll start brand new with that last layer and we'll start feeding it oranges and apples and have it automatically retrain the parameters to figure out what's an apple and what's an orange; and the way that we do that is that for some of those layers, at the beginning we will fix them-- so all the ones in gray, for example, we'll leave unchanged, and the ones to the far right of wherever that dotted line is, we'll let the system change as it starts to learn from those examples of apples and oranges, and eventually you wind up with a system that can distinguish between apples and oranges without having to spend all the time and effort of building all of those parameters from scratch.

So how well does this work? Well, if you've got something that has very close domains, like we were talking about here-- apples and oranges and bananas-- you get pretty good accuracy and it's relatively cheap just to change that last output. From a cybersecurity perspective, however, the adversary has deep knowledge of the teacher. Again, most of the time people are using public models that have already been trained and so, first of all, it's very easy to create adversarial input-- that is, being able to fool the system-- because you know how the system is working on the inside. You know how it's finding edges or finding features or what you have because all those (inaudible) is available to you, and so you can create things that will fool the system by going down paths of the neural network that lead you to the wrong choice.

It's also easier to exfiltrate the model. This is another important cybersecurity feature sometimes called confidentiality, but think in terms of a commercial application where you have a neural net that's looking for bad parts. Pretend you're a car company looking at parts on an assembly line and you want to make sure you have good parts. The neural network that's doing that selection is actually a competitive advantage to other car companies where it can't figure out whether a part is good or not with the same speed and accuracy that the machine learning system can. If they know that it came from this kind of teacher and student arrangement, using transfer learning, by doing very careful inquisition-- that is, sending in sample inputs into the system and seeing how it comes out-- you can actually exfiltrate-- that is figure out, reverse-engineer, what those parameters were without expending the cost of calculating those parameters themselves; and if you're only changing that final layer, it's actually very easy to exfiltrate that model and therefore lose, frankly, intellectual property that you've invested in building that model.

At the other extreme, you can start with all the parameters that came from the original feature model but allow all of the stages to change in response to the new kind of data that you're passing in. So that's better when the domains are not particularly close. You might think this is not a very helpful kind of thing, but it turns out that in practice, even things that you might think are far apart as domains actually wind up having a lot of reusability in these kind of transfer learning systems. You get a much better accuracy when you're just changing the backend layers and it's far more resilient to these teacher-specific attacks that we talked about earlier. But this is really costly to train. Yes, we do save all the initial amounts, but you are still training many, many layers, many, many parameters.

Of course, now we talk about the engineering tradeoffs. So this is where, again, by understanding how these systems work, you can have the discussion with your engineers, with your data scientists about how you want to trade off accuracy and how you want to trade off resiliency. The more layers that you train, the better resiliency you get, but on the other you trade off the amount of work that you have to do in order to get that, and by understanding this is what the data scientist is doing, you could help guide them as to what the most important consideration is among resiliency, accuracy, and cost.

Now, to understand also a little bit about what these adversarial examples actually look like inside the machine model, we're not going to do that by taking apart all those neural networks and looking at the numbers. Instead we're going to take a look at a graphical representation of how these systems work and what it really looks like when you've got one of these kinds of bad inputs or adversarial inputs.

So to do that, we're going to start with the typical way that these systems work, which is you start with the feature space. Again, back to our oranges and bananas. One of those dimensions is the density that we talked about before, that feature that we decided to use earlier, and the other one is color, measured on some particular scale, and then we developed a deep neural network. What came out of the training is a picture that looks like what's on the right. The neural network represents a particular set or subset of the feature space as representing one object or another. So again, for the sake of argument, we're going to say that anything in that top polygon, or the neural network is going to say, that that's a banana; everything in the lower polygon it's going to call an orange. Anything that's not in the polygon (inaudible) doesn't know. And again, you can generalize this. There are lots of variations on this theme, but this is a simple enough example to go to and explain.

The way that you see these actual attacks happen is we get something that's in that feature space, some new picture of an apple or a banana or an orange, and you have an inclusion attack, something that is labeled inside of a particular place, if it falls in that polygon. So it’s not really a banana but the system is going to call it a banana because it fits in that top polygon. Or you have exclusion attack. Put that little sticker on the stop sign and now it no longer looks like a stop sign, or it stops looking like an orange, and it therefore falls out of the orange polygon even though it ought to be in the orange polygon.

Those inclusion and exclusion attacks are frequently what's going on in the examples that we gave, and again, how you-- in practice, these are not two dimensions, there can be thousands of dimensions. These are not sort of strict little kinds of edgy polygons, but nevertheless that's the general idea.

So from that kind of representation, you could see that a way to approach adding resiliency here is to get rid of the inclusion attacks, you cut off all those spikes; and to get rid of those exclusion attacks, you start filling in all those little holes; and that does allow you to, in the abstract sense, be able to manage those particular kinds of attacks.

How do you actually do that? Well, there's a variety of statistical techniques that you can use in order to accomplish either cutting off the spikes or filling in the holes. You could add adversarial examples to the training. Again, going back to that stop sign, you can feed the stop sign with those little pieces of tape on it and tell the training system, "By the way, this is also a stop sign." You can train with a larger subset. It could be that you had a small number of bananas. You start feeding in bananas with spots; you can feed big bananas; you can feed plantains if you think those are bananas, for this kind of discussion; and train with that larger subset.

You could actually just calculate a way to close up that space between the spikes, something called the convex hull of the classification boundary. That could be shown to be effective against particular kinds of attacks that change the system by no more than a particular amount.

Or you can do something called a statistical robust regression. That doesn't mean that this particular regression is effective against cybersecurity attacks. "Statistically robust" refers to the way to manage outliers, but think of it, again, as a way to try to smooth out and fill in some of those peaks.

But all of these methods, in trying to become more resilient, are really trading off against accuracy. The adversarial examples are noisy. When we add in that stop sign with the little tape on it, we've already introduced effectively other kinds of noisy things that the system will now expect to label as stop signs which might not really be a stop sign. We can throw in lots and lots of domain data; you create something called overfitting, and what that does, in terms of the pictures that we just described, creates a lot of raggedy boundaries, and therefore can potentially open up more places that these outliers can fit in.

It could be that your concave boundaries, that sort of hole, are actually legitimate, that it turns out that that kind of density and color really shouldn't be a banana, or it could be that it could be included, that this kind of thing should be in there even though the spike might be an outlier, but it could be that that particular combination of things that spikes up is something that's legitimate. So you can apply ways to try to make that smoother, but you're trading off your kind of accuracy that you can get with resiliency of the underlying model. There is no right answer here. This is all statistics, and as a result, everything is probability, and so you have to make a choice between which of those things is most important to you and what you're willing to pay for it, and that's, again, the discussion that you'd like to have with your data scientist or machine learning expert as they develop these systems if these particular threats are important to you.

By the way, there is an entirely different strategy as well which is used in a lot of safety applications, which is simply redundancy. Have different models and have them effectively vote on what they are seeing in order to see whether you've got an apple or a banana or an orange.

So finally, I'd like to talk about how these problems actually get introduced into the system. So we somewhat abstract said, "We'll change the training data," or change the model builder, or what have you. How does that actually happen? How might you get into these systems in order to start making those kinds of changes?

Well, the key idea here is that machine learning systems are software, and software we know has a whole variety of ways of attack and, as machine learning systems, all of those ways of attack come along with it, and those are the ways that we can get into these systems in order to start making them misbehave.

So for example, what we call coding hygiene. It turns out that the vast majority of problems with software is bad programming-- not checking inputs for validity, arrays that go out of bounds, overflows that aren't checked properties, executable code that gets passed around without validating or containing. All of these are bad programming from a security perspective, and it turns out that machine learning systems have the same problems. So this comes out of Common Vulnerabilities and Exposures database. It's one of the larger databases that you can look at to find problems in software, and this is for TensorFlow, TensorFlow being one of the most popular machine learning based systems that people can use. It suffers from these kinds of problems. How you deal with this is you just need good cyber hygiene. You basically need to have programmers who are building these systems understand how to build secure coding regardless of how talented they are in the statistics part of the calculation.

A second problem is the software supply chain. As we learned from things like transfer learning, you build these systems from parts. You don't program them from scratch. Just like the regular software industry has moved away from building software from scratch, machine learning systems very much are using existing frameworks, using existing data sources, using existing training models as a way to start.

You can think of these as supplies. However, there's no way to validate that the supply you're getting hasn't been altered in some way, that the images you're looking at are legitimate images, that, as we mentioned before, the tool that's building the model doesn't inject something else along the way; and it turns out that it can be very difficult to find.

For example, you can find deep fakes easily throughout the internet and throughout the machine learning community. Most of these are generated for research purposes but you could imagine that an adversary can start generating these deep fake kind of data in more interesting areas that you'd want to look at.

Now a bigger problem is that even if you have deep fakes, there's no real way to determine it. The research level right now in being able to monitor and detect where there's deep fake videos and deep fake images is really quite immature, and so it's very easy to get bad information into a system. Again, there's lots of people working on this, but right now we cannot detect them as fast as we can build them.

Now in general, this notion of software supply chain risk in the software community has been recognized for quite a while and there have been methods used that can be applied to fix this as well. The first is by knowing who your supplier is. For example, a best practice in this whole area is not to let every single developer go onto the internet and download anything they like, but rather have some kind of process within your organization that's a single point of getting this information and being able to validate it, or at least accept it, against whatever criteria the organization's risk is willing to take.

The second is understanding the product security. That is, does the product have problems with it? Simple way to do this is what I said earlier. One can go to the CVE database that's maintained by the government and look to see whether the particular software that you've got has particular vulnerabilities in it or not. If you're more aggressive and you're willing to take the investment, there's a variety of tools out there that can be used to evaluate product security as well.

You should look at product distribution. That is, how did the product come from wherever it originated into your hands? So when we talk about a supplier, it turns out, especially in this area, software passes through many hands, from the original developer into where you got it from, and the question is: Do you trust each of those stages of the distribution chain? And if not, then you should find a different way to get software or the data that you want.

Finally, you want the operational control, and that is you want to make sure that only the way that it's used itself is secure. This could be physical security. We used my example earlier about the obscene chat bot. If, in fact, the only people that are allowed to use it were not people that were going to give it obscene information, then you don't have to specifically change the training model in order to prevent that. If you are going to allow that, then it is something that you have to manage in the operational product control.

Shane McGraw: Hey Mark, this is Shane. Could we work in an audience question?

Mark Sherman: Sure.

Shane McGraw: We got one from Walter asking: If a vulnerability is discovered in a trained network that could be exploited by an adversary, what are the alternatives to retraining and revalidating the network? Can one patch the AI system?

Mark Sherman: To my best experience, the short answer is no, that it's not like fighting a bad pointer in conventional software, saying, "I can just insert a check for the bad pointer." This goes back, again, to the statistical nature of these machine learning systems, that everything is a probability, and so when we say, "Find the bad example," it's true that that bad example worked its way through some confidence interval that got violated or didn't match what you expected. So you do have to change the confidence interval. Ideally you don't have to retrain it from scratch, but by understanding what the range of errors this represents, you can specifically train against that. To give you, again, a somewhat made-up example, one of the techniques we talked about was how you can have the system be resilient against a certain number of pixel changes in an image. So if you have a bunch of images of oranges and an adversary may do things like try to change a couple of pixels, how many pixels can it change without being fooled that it's an orange or not? There's math here which we won't go into, that allows you to say how you can adjust the model so that as we change it, that parameter of how many pixels you're allowed to change, the model can be more robust. If the attack we know is because some number of pixels have been changed and you can somehow manage that, then you can adjust the model specifically to handle that kind of attack. So that's a way by which you can react to it, but (inaudible) keep refraining these are statistical systems; they're not the same kinds of software like payroll systems, which are deterministic.

Shane McGraw: All right, thanks. We got one more from Robert, if we can work it in here, and then we'll let you get back to the content after this one and work some other in later. But from Robert: If resiliency and accuracy are opposed, are you state-- but metrics can be used to measure resiliency, and does this metric also take into account redundancy such as those in ensembles?

Mark Sherman: Sure. So the unsatisfying answer is that there are actually a whole variety of metrics that have been developed. I gave you one of them just now, which is how many pixels have been changed in an image. There are other ones as well, but it's all to effectively measure changes in accuracy; that is, how many-- what percentage of bad things you're going to find, versus how many changes in the input that you're willing to tolerate, and there's different kinds of changes in the input, and there's different kinds of things that you might accept as good or bad, and it's not-- (inaudible) say good or bad, it's, "I can make a confident determination," or "I cannot have a confident determination." So it may be that you don't get fooled into thinking an orange is a banana, but what you wind up saying is, "I can't tell." Like, "Only if I see a perfect orange or a perfect banana, I can recognize it." Anything away from the model of perfection, I just throw up my hands and say, "I don't know what the answer is." And so there's a whole range of-- I do not know of a convergence happening yet, but the literature does have many of those alternatives that are there.

Shane McGraw: All right, we'll let you get back to the content. We've got another couple questions in the queue but we'll save them for next round here.

Mark Sherman: All right. So another attack that can happen to machine learning systems as well is denial-of-service attacks. Again, most machine learning systems when they're deployed are not set up with the view that they're going to have many of the outside world attacking them, or try to use them simultaneously, and overwhelming them. This is very conventional, and again, the response here is also very conventional. There's a variety of what I call network hygiene approaches that are well publicized that you put into your system in order to guard against these kinds of denial-of-service attacks.

Another common place that you get access to these systems to try to change them is where they interconnect with other systems. Machine learning systems are almost never standalone. At least that's been my experience; I think this (inaudible) most people. They always interconnect with other systems; that's what makes them useful. The system needs to somehow understand what it's looking at (inaudible) quality assurance is watching, or just go by for some reason and looking to pick out bananas, well that's connected to some system that's going to pick out bananas, or report how many oranges it's seeing and all sorts of things like that.

It turns out that the new operating environments, that the interconnections between systems has been demonstrated to be one of the two major sources of vulnerabilities in systems because the interconnections generally don't get high fidelity in their handoff between the cybersecurity assumptions that are made on each side of the connection, and therefore they represent the place that attackers can attack.

If you're curious, as I said, that's one of the two-- the other one turns out to be how popular a system is, as the other big criterion as to whether it's really a good target for attack or not, and I guess that's just a recognition that-- to do what the bank robber says. "Why do you rob banks?" "That's where the money is." Why do cybers attack popular systems? Because that's where they can go and get the things that they want. From a technical perspective, it's where these parts interconnect that turns out to be the weakest part of these systems for attack.

We also have to remember that systems are being used by people, and insider threat is especially a pernicious threat in these kinds of machine learning systems. Typically if you have other kinds of systems they may steal the software, they may do some very simple kinds of changes, like being able to look for special certificates so that people who aren't authorized can do authorization. Those are very gross effects that insiders can do-- someone putting a USB into their machine and sucking out all the data and walking out with it and giving it to a competitor. In the case of machine learning, we're seeing that in that model, there are many different subtle places that you can make changes that are actually quite hard to detect to see if someone has actually made that kind of change in order to have the system behave in a way that it shouldn't.

One deals with insider threat, again, using pretty conventional kinds of things, the kind of organizational processes we put in place to manage and monitor all the software and data that you're using to build the system, as well as training people, because it turns out a lot of what we think of as insider threats turns out to be inadvertent, people who don't realize that the procedures they're following, the behaviors that they're demonstrating actually can affect the system availability and system confidentiality.

The last comment I'd like to make about the human element here, I call it the Fake News and Trustworthiness. Ultimately these machine learning systems are used by people to make decisions. Because they are largely opaque-- that is, I don't know really how the system decided that something was a banana or an orange-- I have this sort of gut reaction, which is really simple in this case, that because the density-- bananas have a certain density, oranges have a certain density-- if it's looking for that, then it'll come out the right way. As you get into real systems that are having much more complicated things-- how to set bail, which houses to check for lead-- then it becomes much harder for the people to understand how a system came to a conclusion.

They had this problem in Michigan, where two houses next to each other, the AI system they used said, "This house needs to be checked for lead. That house doesn't need to be checked for lead," and people would say, "I don't understand. It looks identical to me. Why is it making that kind of decision? I can't trust the system to choose when I have to check for lead and when I can't." In fact, it's so pervasive that a recent study done by Gartner said that most lines of businesses, 80 percent of them, don't trust what's coming out of the AI and will actually second-guess what's coming out of the AI.

This is not a short-term problem that can be fixed. We need the further research that can provide the approved explanations as to how did the system come to what it came to, but we also need to have, I think, expectation, which is through education and experience. Again, going back to my refrain, these are statistical systems. They say how probable does it think that it's got an orange, or how probable does it think it's found a banana. It's not like a payroll where the amount of money in your bank account should be a particular number. If it's not that number, then there's an error in the system. When we talk about bananas, we only talk about percentages, and recognizing that machine learning systems are really statistical systems is key to understanding what they produce and when to use them and when not to use them.

So we've gone through an overview of what kinds of attacks look like. We went through what the attack surfaces are; that is the model of building these kinds of systems and where you might touch it that would cause it to misbehave. We looked at a particular example with transfer learning, why that is so appealing, but also the additional risks that come from it. We talked about in general what the adversarial examples look like as far as what the math is doing under the covers, and then we explained a little bit about how one can actually get to those systems to make those changes that we identified as an attack surface.

If you had some more questions, Shane, I guess now would be a good time.

Shane McGraw: Yeah, we do, one from Orlando, Mark, asking: Are there reports on NN-- so I'm assuming neural networks-- that have been exploited, and what was done to mitigate the exploits?

Mark Sherman: So there's a lot of academic literature on this. I'm not aware of any publicly released examples of neural nets that have been attacked and exploited. But again, there is a lot-- if you take a look (inaudible) the IEEE conference on privacy and security, every year there's-- dozens is too large-- certainly handfuls of papers that give another version of attack and another way that they tried to defend against that attack.

Shane McGraw: Okay, next from Michael, asking: You mentioned a mitigation technique for vulnerabilities was redundancy. Given the inherent transferability of the majority of adversarial attacks on ML models.

Mark Sherman: So the idea is that-- remember, part of the way that these systems are constructed is that you use-- and I lost (inaudible)-- but there is the algorithm for generating the neural net. There are lots of different algorithms that can be used for generating the neural net. So when you talk about redundancy, you're using different of those models, not just repeating the same model and perhaps a little bit of different training data. Yes, that will probably wind up-- to the concern raised, which is the bad input may be incorrectly classified by both of those systems, but if you have substantially different model construction, then you make yourself more resilient to those kinds of attacks.

Shane McGraw: A follow-up from Michael was: Is there research focused on the right mix of architecture-slash-models to enable max resiliency and robustness?

Mark Sherman: So at the moment the literature only talks about tradeoffs. So there are papers that will show you curves. Again, we talked about different metrics that you can have for both the accuracy and resiliency, and the papers will show the curves that result from the tradeoffs that you can make between the trustworthy. Whether that gives you an optimal result or not-- I think more of what I'll call a business decision rather than a technical decision as to where on that frontier you think makes good sense for your particular application. Not as satisfying as you would like, but that's sort of what the state of the art is.

Shane McGraw: Gotcha. Next one from William asking: Can a neural network attack be deterministic? Meaning, can an attacker poison the data so that an orange is classified as a banana, or are they just looking to make the system unreliable?

Mark Sherman: No, you can actually-- in fact, you can cause systems to drift, and one of my favorites is something called a (inaudible) frog attack, where by feeding-- especially active learning systems-- by slowing moving the inputs from an orange to a banana, if you like, and calling it an orange, it slowly changes from one to the other, for our made-up example here, you can get the system to start classifying bananas as oranges, and that's been demonstrated, again, in academic systems extensively.

Shane McGraw: All right, from Christian, asking: What is your opinion about the ops practices-- DevOps, MIOps, and those new threats?

Mark Sherman: So, I'm a great fan of DevSecOps, and using them as well to build machine learning-based systems as well. So going back to the generic comment about the supply chain-- we didn't do a full supply chain analysis here. There's actually other talks we have I think on our website that go into that in a whole lot more detail, but the short version is that it's not only, if you like, the raw materials, the training data, that can be corrupted, but the tools as well that can be corrupted. So in your pipeline-- again, in a conventional sense, the compiler here-- the trainer itself can be corrupted and therefore you can attack the whole system by attacking the DevSecOps pipeline and the tools that are being used in that pipeline. I think there might be more behind the question, but yes, those all represent attacks, but our experience has shown that using that kind of agile approach to building systems is still a better way to get at least the features that you want faster. The whole notion of how one adds security into that development process is a whole other talk; and of course if you're using that technology, that pipeline technology, to build AI systems, so too how you add in security there is a whole other talk. But it is an issue. I'm not quite sure how much more (inaudible).

Shane McGraw: Gotcha. Is there any more content you want to get back to, Mark, or do you want to stay here with questions? We got about another three, four, five questions in the queue. We can stay here, or do you have any other final closing thoughts you wanted to work in before we get back to Q&A?

Mark Sherman: No, let's handle some more questions, about the time (inaudible).

Shane McGraw: Right. One from Geo asking: Could explainability or traceability also be under attack for ML, if achievable at all?

Mark Sherman: So my opinion is-- and this is an opinion-- that if we can get explainability, it will depend on, again, some underlying calculations, and some of the early work here looks like that, and where, to use the face example, that-- it's a line. Usually the neural nets are not as simplistic as that example, but you can say, "I saw that because the distance between their eyebrows is a certain amount and the picture I'm looking for is that same amount. Therefore, that's why I'm looking at a picture of Joe." So to the extent that you can do that, yes, you can get some explainability so we have some belief that it measures Joe, but to the earlier part of the attack surface, just like any other part of this whole process, you can attack it by attacking the system that's comparing the picture of-- the distance of the eyebrows on Joe in the picture and the distance of the eyebrows of the sample. If I alter that system, attack that system to always say, "Yes, that's always Joe," or "No, that's never Joe," then yes, I can make the explainability be just as false as the result.

Shane McGraw: Next one asking: Can you please elaborate on how the integration point can be a threat for a machine learning system?

Mark Sherman: Sure. So the short version is that the-- I use the example of the quality assurance thing. As the quality assurance system is feeding in requests saying, "Kick out all oranges," (inaudible) if you can attack that first system so that every time it gets a request to do something, it sends into the machine learning system that, "I want to look at only bananas," then the machine learning system thinks it's behaving correctly, but because what got put into the machine learning system was invalidated by the previous stage, then the whole system has wound up being compromised. So no, you haven't generated the bad stop sign, but you have caused the whole system to no longer behave as expected. In general, the machine learning systems don't have the same expectation of cleaning their data as they come in, and as a result the data that get fed to them from external systems usually are-- without attention to it-- can contain problems in them. Again, I'm using somewhat facetious examples-- your stop sign versus a speed limit sign example. If what's being fed into it is a system that is inherently noisy about how it captures pictures, and so it starts arbitrarily throwing noise onto the picture of the stop signs, the machine learning system thinks that it's getting clean stop sign or speed limit signs when in fact it's getting noisy stop signs, and that's because when it was designed it was designed as sort of one thing; the thing generating the pictures of the stop signs (inaudible) thought it was generating something else, and that mismatch is what causes the attack.

Shane McGraw: Great answer, thank you. Kyle with a comment, and maybe I'm going to let you just speak to the comment. Again, it was from a point you made. He just wanted to reinstate the importance of safe coding. "An adversary who can alter variables-- for example, weights or parameters-- can cause damage without deep knowledge of the model data." So I'll let you comment to that or make another point.

Mark Sherman: So, I agree. First, I would label that more like the insider threat, by changing the parameters, but having someone being able to change the code-- by "insecure code" I'm usually referring to bad code. So in the generated machine model, for example, there's a very common exploit called buffer overflow. If you don't code the machine model correctly, as it's doing its calculations among all those nodes, if it generates a buffer overflow then that represents a way to change the parameter that you just talked about and therefore have the system generate a bad results; whereas I think an insider would go in and start tweaking things.

Shane McGraw: Next we've got one from Heidi: I have read that some financial breaches have occurred from insider threats having access to the data itself. What are some of the best automated methods to target this, in your opinion?

Mark Sherman: So actually I'm going to refer on this one and say I'm aware of a lot of work being done on insider threats within the financial institutions, but I also know that I'm not much of an expert on it and therefore, I'm sorry, but I'll pass on that one.

Shane McGraw: Okay. We can try to get some information on that, Heidi. Another one from Faqeer, asking: As neural networks are non-deterministic, other than evaluating the model multiple iterations, any more better techniques in the literature available to better test the machine learning models?

Mark Sherman: So I wouldn't call them non-deterministic; I would call them statistical. Once you have the model trained, it's as algorithmic as the quadratic formula. You put in the data and the answer will come out the same way every single time. But the way that it picks those parameters is, again, using statistical techniques, and so depending on what subset of data you pick and so forth you'd get a totally different answer, a different model that will have-- those polygons that I showed you earlier will be different shapes. How you'd make that more explainable, if that's what (inaudible), or resilient, that goes back just to the techniques we had for managing the fact that you're dealing with a statistical system and a generation of a statistical system.

Shane McGraw: Okay, we're caught up in the Q&A, Mark, so I'm going to let you wrap it up with any closing thoughts or maybe where you see the future of this, or other research your team may be doing, and then we'll see if we get any other questions in the queue before we sign off right before two thirty.

Mark Sherman: Sure, thank you. So this is a growing area of research. You heard me allude to several times that there are different kinds of attacks and those different kinds of attacks, although they all are of a form of (inaudible) where to fit in those polygons, the way that the models are calculating those polygons is different, and therefore the way that the adversaries generate the bad examples to find those wrong places in the polygon is different as well. We have an unfortunate history that we identify a new way that those polygons can be built or that the way of building them can be attacked, the way that they can be defended, and the history has been over the last decade or so that every time you come up with a way to defend against a specific kind of attack, they found a way to get around that defense. It's taken between a week and a month from when such a protection has been published to when a workaround has been equally published in the literature. And so we are still facing somewhat of an uphill battle for making this a manageable science, and so right now the best we can hope for is understanding the risks and the tradeoffs that you can make in order to try to mitigate some of those risks against particular threats that you might be facing in your machine learning system.

Shane McGraw: Great. So we did get a couple more questions, Mark, and then we'll sign off after these two, asking: Could you talk more about some of the common end-goals of attackers when they attack ML systems?

Mark Sherman: So I think most of the time they are actually looking at active learning systems, because those are the most popular ones, that have public interfaces to them. Private ones usually don't see that kind of thing. Like the Michigan one that talked about lead houses--that was done by the Water Authority and the EPA (inaudible) places that are private. But public ones like the Microsoft chatbot example that I alluded to earlier, those are the ones that people seem to be attacking and we see pretty routinely these drifts, where you start feeding in bad data a little bit at a time, and that seems to be an ongoing behavior out in the market. The attempt there is to try to understand what I'll call fake inputs, that in order to do that kind of drift many times we have to start generating things that have unusual artifacts in them that cause the system to start misbehaving, and so a lot of attention is being given on how one then can detect raw data that has these artifacts in them, raising a flag saying, "This is probably not a legitimate thing. I don't want to start having my system react to this kind of tainted data."

Shane McGraw: All right, last one Mark, and we'll wrap up. This one from Rick, asking: What are your best recommendations to ensure the integrity of source data has not been compromised? Any specific practices, hashes, etcetera?

Mark Sherman: So the best advice we've been able to come up with is based on that SPDO process that we mentioned earlier-- supplier, product distribution, operations-- where you know where your supplier came from. So, for example, Google publishes data; Facebook publishes data. If you're getting it directly from Facebook or you're getting it directly from Google, you can think that that's a reasonable authority. If, however, you get it from Joe's Data Store who says they got it from some other place, got it from the University of We Don't Know Where, who said they got it from Google, now you don't know who's touched it along the way and therefore you don't have any real confidence in the data, and therefore you don't have any trust of the data. So just know your source.

Shane McGraw: Great. Mark, great discussion today. Thanks again for sharing your expertise.

Mark Sherman: Great. Thank you very much, everybody.

Shane McGraw: We wanted to thank everybody for attending today. Upon exiting, please hit the Like button below and share the archive if you found value in today's talk. Also, you can subscribe to the SEI YouTube channel by clicking the SEI seal in the lower right corner of the video window. Lastly, join us for our next webcast, which will be November 17. The topic will be cybersecurity engineering with Carol Woody and Rita Creel, and you'll all be emailed the registration link. Any questions from today's event, please send to info@sei.cmu.edu. Thanks everyone. Have a great afternoon.