AN ARCHITECTURAL RISK ANALYSIS OF MACHINE LEARNING SYSTEMS:
Toward More Secure Machine Learning

Gary McGraw, Ph.D.
Harold Figueroa, Ph.D.
Victor Shepardson
Richie Bonett

Berryville Institute of Machine Learning (BIML)

Version 1.0 (1.13.20)

At BIML, we are interested in "building security in" to machine learning (ML) systems from a security engineering perspective. This means understanding how ML systems are designed for security, teasing out possible security engineering risks, and making such risks explicit. We are also interested in the impact of including an ML system as part of a larger design. Our basic motivating question is: how do we secure ML systems proactively while we are designing and building them? *This architectural risk analysis (ARA) is an important first step in our mission to help engineers and researchers secure ML systems.*

We present a basic ARA of a generic ML system, guided by an understanding of standard ML system components and their interactions.

Securing a modern ML system must involve diving into the engineering and design of the specific ML system itself. This ARA is intended to make that kind of detailed work easier and more consistent by providing a generic baseline and a set of risks to consider.

Why we Need an ML Risk Analysis at the Architectural Level

Twenty-five years ago when the field of software security was in its infancy, much hullabaloo was made over software vulnerabilities and their associated exploits. Hackers busied themselves exposing and exploiting bugs in everyday systems even as those systems were being rapidly migrated to the Internet. The popular press breathlessly covered each exploit. Nobody really concerned themselves with solving the underlying software engineering and configuration problems since finding and fixing the flood of individual bugs seemed like good progress. This hamster-wheel-like process came to be known as "penetrate and patch."

After several years of public bug whack-a-mole and debates over disclosure, it became clear that bad software was at the heart of computer security and that we would do well to figure out how to build secure software.[1:viega] That was twenty years ago at the turn of the millennium. These days, software security is an important part of any progressive security program. To be sure, much work remains to be done in software security, but we really do know what that work should be.

Though ML (and AI in general) has been around even longer than computer security, until very recently not much attention has been paid to the security of ML systems themselves. Over the last few years, a number of spectacular theoretical attacks on ML systems have led to the same kind of breathless popular press coverage that we experienced during the early days of computer security. It all seems strikingly familiar. Exploit a bug, hype things up in the popular press, lather, rinse, repeat.

We need to do better work to secure our ML systems, moving well beyond attack of the day and penetrate and patch towards real security engineering.

In our view at BIML, an architectural risk analysis (ARA) is sorely needed at this stage. An ARA takes a design-level view of a system and teases out systemic risks so that those risks can be properly mitigated and managed as a system is created.[2:mcgraw] Note that in general an ARA is much more concerned with design tradeoffs and solid engineering than it is with particular bugs in a specific system or individual lines of code. In fact, sloppy engineering itself often leads directly security issues of all shapes and sizes. For this reason, we spend some time talking about aspects of robustness and reasonable engineering throughout this document.

Our work at BIML is by no means the first work in securing ML systems. Early work in security and privacy of ML has taken more of an "operations security" tack focused on securing an existing ML system and maintaining its data integrity. For example, in one section of his seminal paper, Nicolas Papernot uses Saltzer and Schroeder's famous security principles[3:saltzer] to provide an operational perspective on ML security.[4:papernot] In our view, Papernot's work only begins to scratch the surface of ML security design.

Following Papernot, we directly addressing Saltzer and Schroeder's security principles from 1972 (as adapted in *Building Secure Software* by Viega and McGraw in 2001) in Part 2. Our treatment of the principles is more directly tied to security engineering than it is to security operations.

Also of note, our work focuses on "security *of* ML" as opposed to "ML *for* security." That is, we focus our attention on helping engineers make sure that their ML system is secure while other work focuses on using ML technology to implement security features. This is an important distinction. In some cases these two distinct practices have been blurred in the literature when they were (confusingly) addressed simultaneously in the same work.[5:barreno] We do what we can to focus all of our attention on the security *of* ML.

Intended Audience

We have a confession to make. We mostly did this work for ourselves in order to organize our own thinking about security engineering and ML. That said, we believe that what we have produced will be useful to three major audience groups: 1) *ML practitioners and engineers* can use this work to understand how security engineering and more specifically the "building security in" mentality can be applied to ML, 2) *security practitioners* can use this work to understand how ML will impact the security of systems they may be asked to secure as well as to understand some of the basic mechanisms of ML, and 3) *ML security people* can use this detailed dive into a security engineering mindset to guide security analysis of specific ML systems.

Document Organization

Part One of this document extensively covers a set of 78 risks that we have identified using a generic ML system as an organizing concept. To start things off, we provide a list of what we consider the top ten risks in ML systems today. Next we discuss a large set of risks associated with each of nine components of a generic ML system. Our intent is for the long list to be a useful guide for security analysis of specific ML systems. Because of that intent, the list is somewhat dauntingly large, but will be useful when practically applied. Next we discuss known ML attacks and present a simple taxonomy associated with our generic ML model. We also briefly cover ML system attack surfaces. We end Part One with treatment of system-wide risks.

Part Two of this document is a treatment of Saltzer and Schroeder's 1972 security principles (as adapted in *Building Secure Software* by Viega and McGraw in 2001).

You are most welcome to skip around while reading this document, maybe even starting with Part Two. We expect Part One will serve as much as a reference document to refer back to as it serves as an exposition.

One last thing before we dive in; this work (and indeed all of security) is just as much about creating resilient and reliable ML systems as it is about security. In our view, security is an emergent property of a system. No system that is unreliable and fragile can be secure. For that reason, a number of the risks we identify and discuss have as much to do with solid engineering as they have to do with thwarting specific attacks.

BIML

PART ONE: ML Security Risks

Picking a Target: Generic ML

ML systems come in a variety of shapes and sizes, and frankly each possible ML design deserves its own specific ARA. For the purposes of this work, we describe a generic ML system in terms of its constituent components and work through that generic system ferreting out risks. The idea driving us is that risks that apply to this generic ML system will almost certainly apply in any specific ML system. By starting with our ARA, an ML system engineer concerned about security can get a jump start on determining risks in their specific system.

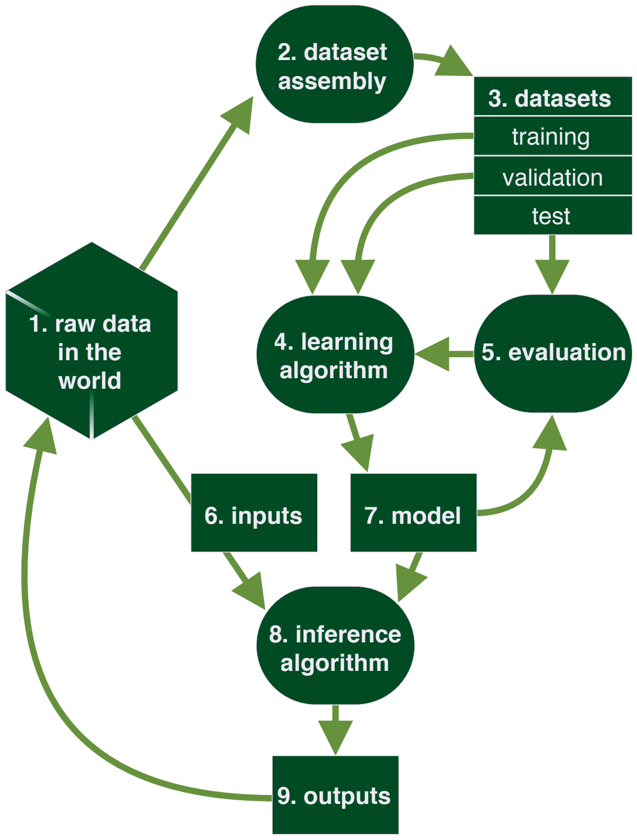

Figure 1: Components of a generic ML system. Arrows represent information flow.

Figure 1 above shows how we choose to represent a generic ML system. We describe nine basic components that align with various steps in setting up, training, and fielding an ML system: 1) raw data in the world, 2) dataset assembly, 3) datasets, 4) learning algorithm, 5) evaluation, 6) inputs, 7) model, 8) inference algorithm, and 9) outputs. Note that in our generic model, both processes and collections are treated as components. Processes—that is, components 2, 4, 5, and 8—are represented by ovals, whereas things and collections of things—that is, components 1, 3, 6, 7, and 9—are represented as rectangles.

The nine components of our generic ML model map in a straightforward way into specific ML models. As an example of this kind of mapping, consider Google's Neural Machine Translation model (GNMT).[6:wu] Here is how that mapping works:

1. *Raw data in the world.* GNMT makes use of numerous Google internal datasets for training; the sources of these data are not made crystal clear, but Google explicitly mentions Wikipedia articles and news sites.
2. *Dataset assembly.* Raw text is organized into sentence pairs between two languages. Sentences are segmented by a model which splits individual words into smaller wordpieces and adds special characters to the beginning of each word. (This is the best performing option proposed; they also evaluate on word-based, character-based, and a mixed model which only splits out-of-vocabulary words into a character representation.)
3. *Datasets.* The parsed text pairs are separated into a training set and test set.
4. *Learning algorithm.* At a high level, GNMT's learning algorithm consists of an Encoder Recurrent Neural Network (RNN), an attention module, and a Decoder RNN.
5. *Evaluation.* The networks are trained by first applying a maximum-likelihood objective until log perplexity converges, and then refined with reinforcement learning. The process continues until the model produces consistent BLEU scores for the test set. (BLEU (an acronym for bilingual evaluation understudy) is an algorithm for evaluating the quality of machine-translated text that has become a *de facto* standard.)
6. *Inputs.* Input to the inference algorithm consists of textual sentences in a particular source language.
7. *Model.* The trained model includes numerous configured hyperparameters and millions of learned parameters.
8. *Inference algorithm.* GNMT is made accessible through an interface that everyone knows as Google Translate.
9. *Outputs.* Outputs consist of textual sentences in the target language.

Given a specific mapping like this, performing a risk analysis by considering the ML security risks associated with each component is a straightforward exercise that should yield fruit.

The Top Ten ML Security Risks

After identifying risks in each component which we describe in detail below, we considered the system as a whole and identified what we believe are the top ten ML security risks. These risks come in two relatively distinct flavors, both equally valid: some are risks associated with the *intentional actions of an attacker*; others are risks associated with an *intrinsic design flaw*. Intrinsic design flaws emerge when engineers with good intentions screw things up. Of course, attackers can also go after intrinsic design flaws complicating the situation.

The top ten ML security risks are briefly introduced and discussed here.

1. **Adversarial examples.** Probably the most commonly discussed attacks against machine learning have come to be known as *adversarial examples*. The basic idea is to fool a machine learning system by providing malicious input often involving very small perturbations that cause the system to make a false prediction or categorization. Though coverage and resulting attention might be disproportionately large, swamping out other important ML risks, adversarial examples are very much real. See [input:1:adversarial examples] below.

2. **Data poisoning.** Data play an outsized role in the security of an ML system. That's because an ML system learns to do what it does directly from data. If an attacker can intentionally manipulate the data being used by an ML system in a coordinated fashion, the entire system can be compromised. Data poisoning attacks require special attention. In particular, ML engineers should consider what fraction of the training data an attacker can control and to what extent. See [data:1:poisoning] below.
3. **Online system manipulation.** An ML system is said to be "online" when it continues to learn during operational use, modifying its behavior over time. In this case a clever attacker can nudge the still-learning system in the wrong direction on purpose through system input and slowly "retrain" the ML system to do the wrong thing. Note that such an attack can be both subtle and reasonably easy to carry out. This risk is complex, demanding that ML engineers consider data provenance, algorithm choice, and system operations in order to properly address it. See [alg:1:online], [inference:1:online], and [data:7:online] below.
4. **Transfer learning attack.** In many cases in the real world, ML systems are constructed by taking advantage of an already-trained base model which is then fine-tuned to carry out a more specific task. A data transfer attack takes place when the base system is compromised (or otherwise unsuitable), making unanticipated behavior defined by the attacker possible. See [data:2:transfer], [model:1:improper re-use] and [model:2:Trojan] below.
5. **Data confidentiality.** Data protection is difficult enough without throwing ML into the mix. One unique challenge in ML is protecting sensitive or confidential data that, through training, are built right into a model. Subtle but effective extraction attacks against an ML system's data are an important category of risk. See [raw:1:data confidentiality] below.
6. **Data trustworthiness.** Because data play an outsize role in ML security, considering data provenance and integrity is essential. Are the data suitable and of high enough quality to support ML? Are sensors reliable? How is data integrity preserved? Understanding the nature of ML system data sources (both during training and during execution) is of critical importance. Data borne risks are particularly hairy when it comes to public data sources (which might be manipulated or poisoned) and online models. See [raw:2:trustworthiness] below.
7. **Reproducibility.** When science and engineering are sloppy, everyone suffers. Unfortunately, because of its inherent inscrutability and the hyper-rapid growth of the field, ML system results are often under-reported, poorly described, and otherwise impossible to reproduce. When a system can't be reproduced and nobody notices, bad things can happen. See [alg:2:reproducibility] below.
8. **Overfitting.** ML systems are often very powerful. Sometimes they can be too powerful for their own good. When an ML system "memorizes" its training data set, it will not generalize to new data, and is said to be overfitting. Overfit models are particularly easy to attack. Keep in mind that overfitting is possible in concert with online system manipulation and may happen while a system is running. See [eval:1:overfitting] below.
9. **Encoding integrity.** Data are often encoded, filtered, re-represented, and otherwise processed before use in an ML system (in most cases by a human engineering group). Encoding integrity issues can bias a model in interesting and disturbing ways. For example, encodings that include metadata may allow an ML model to "solve" a categorization problem by overemphasizing the metadata and ignoring the real categorization problem. See both [assembly:1:encoding integrity], [raw:5:encoding integrity], and [raw:10:metadata] below.

10. **Output integrity.** If an attacker can interpose between an ML system and the world, a direct attack on output may be possible. The inscrutability of ML operations (that is, not really understanding how they do what they do) may make an output integrity attack that much easier since an anomaly may be harder to spot. See [output:1:direct] below.

Risks in ML System Components

In this section we identify and rank risks found in each of the nine components of the generic ML system introduced above. Each risk is labeled with an identifier as follows: [<component label>:<risk number>:<descriptor>]. We use these labels to cross-reference risks and as shorthand pointers in the rest of the document. After each component's list of risks are a set of controls, some associated with particular risks and others generic.

1. **Raw data in the world risks:** If we have learned only one thing about ML security over the last few months, it is that data play just as important role in ML system security as the learning algorithm and any technical deployment details. In fact, we'll go out on a limb and state for the record that we believe *data make up the most important aspects of a system to consider when it comes to securing an ML system.*

   Our usage of the term *raw data* in this section is all inclusive, and is not limited to training data (which for what it's worth is usually created from raw data). There is lots of other data in an ML system, including model parameters, test inputs, and operational data.

   Data security is, of course, a non-trivial undertaking in its own right, and all collections of data in an ML system are subject to the usual data security challenges (plus some new ones).

   Eventually, a fully-trained ML system (whether online or offline) will be presented with new input data during operations. These data must also be considered carefully during system design.

   [raw:1:data confidentiality]
   Preserving data confidentiality in an ML system is more challenging than in a standard computing situation. That's because an ML system that is trained up on confidential or sensitive data will have some aspects of those data built right into it through training. Attacks to extract sensitive and confidential information from ML systems (indirectly through normal use) are well known.[7:shokri] Note that even sub-symbolic "feature" extraction may be useful since that can be used to hone adversarial input attacks.[4:papernot]

   [raw:2:trustworthiness]
   Data sources are not trustworthy, suitable, and reliable. How might an attacker tamper with or otherwise poison raw input data? What happens if input drifts, changes, or disappears?[8:barreno]

   [raw:3:storage]
   Data are stored and managed in an insecure fashion. Who has access to the data pool, and why? Access controls can help mitigate this risk, but such controls are not really feasible when utilizing public data sources. This kind of risk brings to mind early attempts to create mathematically random data for cryptographic security through combining sets of inputs that could ultimately be influenced by an attacker (such as process id, network packet arrival time, and so on). Needless to say, entropy pools controlled by an attacker are low entropy indeed. Ask yourself what happens when an attacker controls your data sources.

   [raw:4:legal]
   Note that public data sources may include data that are in some way legally encumbered. An obvious example is copyrighted material that gets sucked up in a data stream. Another more insidious example is child pornography which is never legal. A third, and one of the most interesting

legal issues now is that there may be legal requirements to "delete" data (e.g., from a GDPR request). What it means to "delete" data from a trained model is challenging to carry out (short of retraining the model from scratch from a data set with the deleted data removed, but that is expensive and often infeasible). Note that through the learning process, input data are always encoded in some way in the model itself during training. That means the internal representation developed by the model during learning (say, thresholds and weights) may end up being legally encumbered as well.

[raw:5:encoding integrity]
Raw data are not representative of the problem you are trying to solve with ML. Is your sampling capability lossy? Are there ethical or moral implications built into your raw data (*e.g.*, racist or xenophobic implications can be trained right into some facial recognition systems if data sets are poorly designed)?[9:phillips]

[raw:6:representation]
Representation plays a critical role in input to an ML system. Carefully consider representation schemes, especially in cases of text, video, API, and sensors. Is your representation rich enough to do what you want it to do? For example, many encodings of images are compressed in a lossy manner. This will impact your model, figure out how.

[raw:7:text encoding]
Text representation schemes are not all the same. If your system is counting on ASCII and it gets Unicode, what happens? Will your system recognize the incorrect encoding and fail gracefully or will it fail hard due to a misinterpreted mismatch?

[raw:8:looping]
Model confounded by subtle feedback loops. If data output from the model are later used as input back into the same model, what happens? Note that this is rumored to have happened to Google translate in the early days when translations of pages made by the machine were used to train the machine itself. Hilarity ensued. To this day, Google restricts some translated search results through its own policies.

[raw:9:data entanglement]
Entangled data risk. Always note what data are meant to represent and be cognizant of data entanglement. For example, consider what happens if a public data source (or even an internal source from another project) decides to recode their representation or feature set. Note that "false features" can also present an entanglement problem as the famous husky-versus-wolf classifier demonstrated by acting (incorrectly) as a snow detector instead of a species detector. Know which parts of your data can change and which should not ever change.[10:sculley]

[raw:10:metadata]
Metadata may help or hurt an ML model. Make note of metadata included in a raw input dataset. Metadata may be a "hazardous feature" which appears useful on the face of it, but actually degrades generalization. Metadata may also be open to tampering attacks that can confuse an ML model.[11:ribeiro] More information is not always helpful and metadata may harbor spurious correlations. Consider this example: we might hope to boost performance of our image classifier by including exif data from the camera. But what if it turns out our training data images of dogs are all high resolution stock photos but our images of cats are mostly facebook memes? Our model will probably wind up making decisions based on metadata rather than content.

[raw:11:time]
If time matters in your ML model, consider time of data arrival a risk. Network lag is something easily controlled by an attacker. Plan around it.

[raw:12:sensor]
Always consider the technical source of input, including whether the expected input will always be available. Is the sensor you are counting on reliable? Sensor blinding attacks are one example of a risk faced by poorly designed input gathering systems. Note that consistent feature identification related to sensors is likely to require human calibration.

[raw:13:utility]
If your data are poorly chosen or your model choice is poor, you may reach incorrect conclusions regarding your ML approach. Make sure your methods match your data and your data are properly vetted and monitored. Remember that ML systems can fail just as much due to data problems as due to poorly chosen or implemented algorithms, hyperparameters, and other technical system issues.

Associated controls. Note that the labels refer to the original risks (above) which have controls that may help alleviate some of the risk directly:

[raw:generic] Protect your data sources if you can.

[raw:generic] Sanity check your data algorithmically before you feed it into your model (e.g., using outlier detection, mismatched unit discovery, data range distribution analysis, and so on). For example, make sure that your data properly characterize and represent the problem space so that the ML model learns what it is supposed to learn. Ironically, this is one of the most difficult engineering problems involved in ML as a field.

[raw:generic] Transform your data to preserve data integrity. This might even involve cryptographic protection.

[raw:generic] Featurize your data so that it is consistently represented. Note that this cuts against the grain of some aspects of "deep learning" (mostly because it turns out to be an exercise for the humans), but may result in a more robust ML system. The tension here is a classic issue in ML. Humans are almost always in the loop, carefully massaging data and setting up the problem and the technology to solve the problem. But at the same time the tendency to let an ML system "magically" do its work is often over-emphasized. Finding the right balance is tricky and important.

[raw:generic] Use version control technology to manage your datasets. Carefully track change logs, diffs, etc, especially when it comes to large datasets.

[raw:1:data confidentiality] Design your ML system so that data extraction from the model is expensive. Consider whether there are mathematical properties of the raw input space that lend themselves to particular models (let that help guide choice of model).

[raw:6:representation] Manual review of representation and periodic validation is a good thing. Consider what is thrown out or approximated (sometimes for computational reasons) in your data representation and account for that.

[raw:8:looping] Look for loops in data streams and avoid them. If raw data come from public sources and system output is also made public, loops may arise without your awareness. As an example, consider what happens when a machine translation system starts using its own translations as training data (as once happened to Google Translate).

[raw:12:sensor] Sensor risks can be mitigated with correlated and overlapping sensors that build and maintain a redundant data stream.

2. **Dataset assembly risks:** In order to be processed by a learning algorithm, raw input data often must be transformed into a representational form that can be used by the machine learning algorithm. This "pre-processing" step by its very nature impacts the security of an ML system since data play such an essential security role.

   Of special note in this component is the discrepancy between online models and offline models (that is, models that are continuously trained and models that are trained once and "set"). Risks in online models drift, and risks in offline models impact confidentiality.

   [assembly:1:encoding integrity]
   Encoding integrity issues noted in [raw:5:encoding integrity] can be both introduced and exacerbated during pre-processing. Does the pre-processing step itself introduce security problems? Bias in raw data processing can impact ethical and moral implications. Normalization of Unicode to ASCII may introduce problems when encoding, for example, Spanish improperly, losing diacritics and accent marks.

   [assembly:2:annotation]
   The way data are "tagged and bagged" (or annotated into features) can be directly attacked, introducing attacker bias into a system. An ML system trained up on examples that are too specific will not be able to generalize well. Much of the human engineering time that goes into ML is spent cleaning, deleting, aggregating, organizing, and just all-out manipulating the data so that it can be consumed by an ML algorithm.

   [assembly:3:normalize]
   Normalization changes the nature of raw data, and may do so to such an extent that the normalized data become exceedingly biased. One example might be an ML system that appears to carry out a complex real-world task, but actually is doing something much easier with normalized data. Destroying the feature of interest in a dataset may make it impossible to learn a viable solution.

   [assembly:4:partitioning]
   When building datasets for training, validation, and testing (all distinct types of data used in ML systems), care must be taken not to create bad data partitions. This may include analysis of and comparisons between subsets to ensure the ML system will behave as desired.

   [assembly:5:fusion]
   Input from multiple sensors can in some cases help make an ML system more robust. However, note that how the learning algorithm chooses to treat a sensor may be surprising. One of the major challenges in ML is understanding how a "deep learning" system carries out its task. Data sensitivity is a big risk and should be carefully monitored when it comes to sensors placed in the real world.

   [assembly:6:filter]
   An attacker who knows how a raw data filtration scheme is set up may be able to leverage that knowledge into malicious input later in system deployment.

   [assembly:7:adversarial partitions]
   If an attacker can influence the partitioning of datasets used in training and evaluation, they can in some sense practice mind control on the ML system as a whole. It is important that datasets reflect the reality the ML system designers are shooting for. Boosting an error rate in a sub-category might be one interesting attack. Because some deep learning ML systems are "opaque," setting up special trigger conditions as an attacker may be more easily accomplished through manipulation of datasets than through other means.[8:barreno]

[assembly:8:random]
Randomness plays an important role in stochastic systems. An ML system that is depending on Monte Carlo randomness to work properly may be derailed by not-really-random "randomness." Use of cryptographic randomness sources is encouraged. "Random" generation of dataset partitions may be at risk if the source of randomness is easy to control by an attacker interested in data poisoning.

Associated controls. Note that the labels refer to the original risks (above) which have controls that may help alleviate some of the risk directly:

[assembly:generic] Provide data sanity checks that look at boundaries, ranges, probabilities, and other aspects of data to find anomalies before they are included in critical datasets. Consider, for example, signal to noise ratio and make sure that it is consistent enough to include data as they are assembled.

[assembly:5:fusion] Determine how dirty data from a sensor may become, and control for both that and for sensor failure. Using multiple sensors may help, especially if they are not exactly the same kind of sensor or modality.

3. **Datasets risks:** Assembled data must be grouped into a training set, a validation set, and a testing set. The training set is used as input to the learning algorithm. The validation set is used to tune hyperparameters and to monitor the learning algorithm for overfitting. The test set is used after learning is complete to evaluate performance. Special care must be taken when creating these groupings in order to avoid predisposing the ML algorithm to future attacks (see [assembly:7:adversarial partitions]). In particular, the training set deeply influences an ML system's future behavior. Attacking an ML system through the training set is one of the most obvious ways to throw a monkey wrench into the works.

[data:1:poisoning]
All of the first three components in our generic model (raw data in the world, dataset assembly, and datasets) are subject to poisoning attacks whereby an attacker intentionally manipulates data in any or all of the three first components, possibly in a coordinated fashion, to cause ML training to go awry. In some sense, this is a risk related both to data sensitivity and to the fact that the data themselves carry so much of the water in an ML system. Data poisoning attacks require special attention. In particular, ML engineers should consider what fraction of the training data an attacker can control and to what extent.[12:alfeld]

[data:2:transfer]
Many ML systems are constructed by tuning an already trained base model so that its somewhat generic capabilities are fine-tuned with a round of specialized training. A transfer attack presents an important risk in this situation. In cases where the pretrained model is widely available, an attacker may be able to devise attacks using it that will be robust enough to succeed against your (unavailable to the attacker) tuned task-specific model. You should also consider whether the ML system you are fine-tuning could possibly be a Trojan that includes sneaky ML behavior that is unanticipated.[13:mcgraw]

[data:3:disimilarity]
If training, validation, and test sets are not "the same" from a data integrity, trustworthiness, and mathematical perspective, an ML model may appear to be doing something that it is not. For

example, an ML system trained up on six categories but only tested against two of the six may not ultimately be exhibiting proper behavior when it is fielded. More subtly, if an evaluation set is too similar to the training set, overfitting may be a risk. By contrast, when the evaluation set is too different from the eventual future inputs during operations, then it will not measure true performance. Barreno *et al* say it best when they say, "Analyzing and strengthening learning methods in the face of a broken stationarity assumption is the crux of the secure learning problem."[8:barreno]

[data:4:storage]
As in [raw:3:storage], data may be stored and managed in an insecure fashion. Who has access to the data pool, and why? Think about [system:8:insider] when working on storage.

[data:5:dataset weak rep]
Assembling a dataset involves doing some thinking and observation about the resulting representation inside the ML model. Robust representations result in fluid categorization behavior, proper generalization, and non-susceptibility to adversarial input. As an example, a topic model trained on predominantly English input with a tiny bit of Spanish will group all Spanish topics into one uniform cluster (globbing all Spanish stuff together).

[data:6:supervisor]
Some learning systems are "supervised" in a sense that the target result is known by the system designers (and labeled training data are available). Malicious introduction of misleading supervision would cause an ML system to be incorrectly trained. For example, a malicious supervisor might determine that each "tank" in a satellite photo is counted as two tanks. (See also [assembly:2:annotation].)

[data:7:online]
Real time data set manipulation can be particularly tricky in an online network where an attacker can slowly "retrain" the ML system to do the wrong thing by intentionally shifting the overall data set in certain directions.

Associated controls. Note that the labels refer to the original risks (above) which have controls that may help alleviate some of the risk directly:

[data:generic] Try to characterize the statistical overlap between validation and training sets. What is best? Document your decisions.

[data:4:disimilarity] Ensure data similarity between the three datasets using mathematical methods. Just as in software engineering, where "coding to the test" can lead to robustness issues, poor training, testing, and validation data hygiene can seriously damage an ML system.

---

4. **Learning Algorithm risks:** In our view, though a learning algorithm lies at the technical heart of each ML system, the algorithm itself presents far less of a security risk than the data used to train, test, and eventually operate the ML system. That said, risks remain that are worthy of note.

Learning algorithms come in two flavors, and the choice of one or the other makes a big difference from a security perspective. ML systems that are trained up, "frozen," and then operated using new data on the frozen trained system are called *offline systems*. Most common ML systems (especially classifiers) operate in an offline fashion. By contrast, *online systems* operate in a continuous learning mode. There is some advantage from a security perspective to an offline system because the online stance increases exposure to a number of data borne vulnerabilities over a longer period of time.

[alg:1:online]
An online learning system that continues to adjust its learning during operations may drift from its intended operational use case. Clever attackers can nudge an online learning system in the wrong direction on purpose.

[alg:2:reproducibility]
ML work has a tendency to be sloppily reported. Results that can't be reproduced may lead to overconfidence in a particular ML system to perform as desired. Often, critical details are missing from the description of a reported model. Also, results tend to be very fragile—often running a training process on a different GPU (even one that is supposed to be spec-identical) can produce dramatically different results. In academic work, there is often a tendency to tweak the authors' system until it outperforms the "baseline" (which doesn't benefit from similar tweaking), resulting in misleading conclusions that make people think a particular idea is actually good when it wasn't actually improving over simpler, earlier method.

[alg:3:exploit-v-explore]
Part of the challenge of tuning an ML system during the research process is understanding the search space being explored and choosing the right model architecture (and algorithm) to use and the right parameters for the algorithm itself. Thinking carefully about problem space exploration versus space exploitation will lead to a more robust model that is harder to attack. Pick your algorithm with care. As an example, consider whether your system has an over-reliance on gradients and may possibly benefit from random restarting or evolutionary learning.

[alg:4:randomness]
Randomness has a long and important history in security. In particular, Monte Carlo randomness versus cryptographic randomness is a concern. When it comes to ML, setting weights and thresholds "randomly" must be done with care. Many pseudo-random number generators (PRNG) are not suitable for use. PRNG loops can really damage system behavior during learning. Cryptographic randomness directly intersects with ML when it comes to differential privacy. Using the wrong sort of random number generator can lead to subtle security problems.

[alg:5:blind spots]
All ML learning algorithms may have blind spots. These blind spots may open an ML system up to easier attack through techniques that include adversarial examples.

[alg:6:confidentiality]
Some algorithms may be unsuited for processing confidential information. For example, using a non-parametric method like k-nearest neighbors in a situation with sensitive medical records is probably a bad idea since exemplars will have to be stored on production servers. Algorithmic leakage is an issue that should be considered carefully.[4:papernot]

[alg:7:noise]
Noise is both friend and foe in an ML system. For some problems, raw data input need to be condensed and compacted (de-noised). For others, the addition of Gaussian noise during pre-processing can enhance an ML system's generalization behavior. Getting this right involves careful thinking about data structure that is both explicit and well documented. Amenability to certain kinds of adversarial input attack is directly linked to this risk.[14:goodfellow]

[alg:8:oscillation]
An ML system may end up oscillating and not properly converging if, for example, it is using gradient descent in a space where the gradient is misleading.

[alg:9:hyperparameters]
One of the challenges in the ML literature is an over-reliance on "empirical" experiments to determine model parameters and an under-reliance on understanding why an ML system actually does what it does. ML systems have a number of hyperparameters, including, for example, learning rate and momentum in a gradient descent system. These parameters are those model settings not updated during learning (you can think of them as model configuration settings). Setting and tuning hyperparameters is somewhat of a black art subject to attacker influence. If an attacker can twiddle hyperparameters (tweaking, hiding, or even introducing them), bad things will happen. (Also see [inference:3:hyperparameters].)

[alg:10:hyperparameter sensitivity]
Oversensitive hyperparameters are riskier hyperparameters, especially if they are not locked in. Sensitive hyperparameters not rigorously evaluated and explored can give you a weird kind of overfitting. For example, one specific risk is that experiments may not be sufficient to choose good hyperparameters. Hyperparameters can be a vector for accidental overfitting. In addition, hard to detect changes to hyperparameters would make an ideal insider attack.

[alg:11:parameters]
In the case of transfer learning (see [data:2:transfer]) an attacker may intentionally post or ship or otherwise cause a target to use incorrect settings in a public model. Because of the open nature of ML algorithm and parameter sharing, this risk is particularly acute among ML practitioners who naively think "nobody would ever do that."

Associated controls. Note that the labels refer to the original risks (above) which have controls that may help alleviate some of the risk directly:

[alg:4:randomness] Have a security person take a look at your use of randomness, even if it seems innocuous.

[alg:5:blind spots] Representational robustness (for example word2vec encoding in an NLP system versus one-shot encoding) can help combat some blind spot risks.

[alg:6:confidentiality] Know explicitly how the algorithm you are using works. Make sure that your choice preserves representational integrity.

[alg:6:confidentiality] Keep a history of queries to your system in a log and review the log to make sure your system is not unintentionally leaking confidential information. Be careful how you store the log— logging everything can itself introduce a big privacy risk.

[alg:9:hyperparameters] Carefully choose hyperparameters and make notes as to why they are set the way they are. Lock in hyperparameters so that they are not subject to change.

[alg:10:hyperparameter sensitivity] Perform a sensitivity analysis on the set of hyperparameters you have chosen.

5. **Evaluation risks:** Determining whether an ML system that has been fully trained is actually doing what the designers want it to do is a thing. Evaluation data are used to try to understand how well a trained ML system can perform its assigned task (post learning). Recall our comments above about the important role that stationarity assumptions have in securing ML systems.

   [eval:1:overfitting]
   A sufficiently powerful machine is capable of learning its training data set so well that it essentially builds a lookup table. This can be likened to memorizing its training data. The unfortunate side effect of "perfect" learning like this is an inability to generalize outside of the training set. Overfit models can be pretty easy to attack through input since adversarial examples need only be a short distance away in input space from training examples. Note that generative models can suffer from overfitting too, but the phenomenon may be much harder to spot. Also note that overfitting is also possible in concert with [data:6:online].

   [eval:2:bad eval data]
   Evaluation is tricky, and an evaluation data set must be designed and used with care. A bad evaluation data set that doesn't reflect the data it will see in production can mislead a researcher into thinking everything is working even when it's not. Evaluation sets can also be too small or too similar to the training data to be useful.[8:barreno] For more, see Luke Oakden-Rayner's blog entry "AI competitions don't produce useful models" at https://lukeoakdenrayner.wordpress.com/ (accessed 10.8.19).

   [eval:3:cooking the books]
   In some cases, evaluation data may be intentionally structured to make everything look great even when it's not.

   [eval:4:science]
   Common sense evaluation and rigorous evaluation are not always the same thing. For example, evaluation of an NLP system may rely on "bags of words" instead of a more qualitative structural evaluation.[15:reiter]

   [eval:5:catastrophic forgetting]
   Just as data play a key role in ML systems, representation of those data in the learned network is important. When a model is crammed too full of overlapping information, it may suffer from catastrophic forgetting. This risk was much more apparent in the early '90s when networks (and the CPUs they ran on) were much smaller. However, even a large network may be subject to this problem. Online systems are, by design, more susceptible.

   [eval:6:choking on big data]
   In an online model, the external data set available may be so vast that the ML system is simply overwhelmed. That is, the algorithm may not scale in performance from the data it learned on to real data. In online situations the rate at which data comes into the model may not align with the rate of anticipated data arrival. This can lead to both outright ML system failure and to a system that "chases its own tail."

   [eval:7:data problems]
   Upstream attacks against data make training and its subsequent evaluation difficult.

   Associated controls. Note that the labels refer to the original risks (above) which have controls that may help alleviate some of the risk directly:

   [eval:2:bad eval data] Public data sets with well-known error rates (or generalization rates) may combat or help control this risk.

[eval:2:bad eval data] and *[eval:3:cooking the books]* are much harder to pull off when the evaluation data and results are public. The research literature is beginning to move toward reproducible results though release of all ML system code and data.

6. **Input risks:** When a fully trained model is put into production, a number of risks must be considered. Probably the most important set of these operations/production risks revolves around input data fed into the trained model. Of course, by design these input data will likely be structured and pre-processed similarly to the training data. Many of the risks identified above (see especially *raw data in the world* risks and *data assembly* risks) apply to model input almost directly.

   [input:1:adversarial examples] One of the most important categories of computer security risks is malicious input. The ML version of malicious input has come to be known as adversarial examples. While important, adversarial examples have received so much attention that they swamp out all other risks in most people's imagination.[16:yuan]

   [input:2:controlled input stream] A trained ML system that takes as its input data from outside may be purposefully manipulated by an attacker. To think about why anybody would bother to do this, consider that the attacker may be someone under scrutiny by an ML algorithm (a loan seeker, a political dissident, a person to be authenticated, etc).

   [input:3:dirty input] The real world is noisy and messy. Input data sets that are dirty enough will be hard to process. A malicious adversary can leverage this susceptibility by simply adding noise to the world.

   [input:4:looped input] If system output feeds back into the real world there is some risk that it may find its way back into input causing a feedback loop. Sometimes this even happens with ML output data feeding back into training data.

   [input:5:pre-processing replication] The same care that goes into *data assembly* (component 2) should be given to input, even in an online situation. This may be difficult for a number of reasons

   Associated controls. Note that the labels refer to the original risks (above) which have controls that may help alleviate some of the risk directly:

   [input:2:controlled input stream] A multi-modal input stream will be harder to completely control. One way to carry this out might be to use multiple sensors that are not similarly designed or that don't have the same engineering failure conditions.

   [input:3:dirty input] Sanity checks, filters, and data cleaning can control this risk. Of course, those mechanisms can be attacked as well. Note that often pre-processing ends up being more about making an ML system be able to learn than it is about "getting it right."

7. **Model risks:** When a fully trained model is put into production, a number of important risks crop up. Note that some of the risks discussed in the *evaluation risks* section above apply directly in this section as well (for example, [eval:1:overfitting] and [eval:4:catastrophic forgetting] both apply).

   [model:1:improper re-use] ML-systems are re-used intentionally in transfer situations. The risk of transfer outside of intended use applies. Groups posting models for transfer would do well to precisely describe exactly what their systems do and how they control the risks in this document.

[model:2:Trojan] Model transfer leads to the possibility that what is being reused may be a Trojaned (or otherwise damaged) version of the model being sought out.[7:shokri]

[model:3:representation fludity] ML is appealing exactly because it flies in the face of brittle symbolic AI systems. When a model generalizes from some examples, it builds up a somewhat fluid representation if all goes well. The real trick is determining how much fluidity is too much. Representation issues are some of the most difficult issues in ML, both in terms of primary input representation and in terms of internal representation and encoding. Striking a balance between generalization and specificity is the key to making ML useful.

[model:4:training set reveal] Most ML algorithms learn a great deal about input, some of which is possibly sensitive (see [raw:1:data confidentiality]), and store a representation internally that may include sensitive information. Algorithm choice can help control this risk, but be aware of the output your model produces and how it may reveal sensitive aspects of its training data. When it comes to sensitive data, one promising approach in privacy-preserving ML is differential privacy which we discuss below.

[model:5:steal the box] Training up an ML system is not free. Stealing ML system knowledge is possible through direct input/output observation. This is akin to reversing the model.

Associated controls. Note that the labels refer to the original risks (above) which have controls that may help alleviate some of the risk directly:

[model:5:steal the box] Watch the output that you provide (it can and will be used against you).

8. **Inference algorithm risks:** When a fully trained model is usually put into production, a number of important risks must be considered. These encompass data fed to the model during operations (see raw data risks and pre-processing risks), risks inherent in the production model, and output risks.

   [inference:1:online] A fielded model operating in an online system (that is, still learning) can be pushed past its boundaries. An attacker may be able to carry this out quite easily.

   [inference:2:inscrutability] In far too many cases, an ML system is fielded without a real understanding of how it works or why it does what it does. Integrating an ML system that "just works" into a larger system that then relies on the ML system to perform properly is a very real risk.

   [inference:3:hyperparameters] Inference algorithms have hyperparameters, for example sampling temperature in a generative model. If an attacker can surreptitiously modulate the hyperparameters for the inference algorithm after the evaluation process is complete, they can control the system's behavior. (Also see [alg:9:hyperparameters].)

   [inference:3:confidence scores] In many cases, confidence scores (which are paired with classification category answers) can help an attacker. If an ML system is not confident about its answer and says so, that provides feedback to an attacker with regards to how to tweak input to make the system misbehave. Conversely, a system that doesn't return confidence scores is much harder to use correctly (and may be used idiotically). Care should be taken as to what kind of output feedback a user can and should get.

   [inference:4:hosting] Many ML systems are run on hosted, remote servers. Care must be taken to protect these machines against ML-related attacks (not to mention the usual pile of computer security stuff).

   [inference:5:user risk] When a user decides to use an ML system that is remote, they expose their interests (and possibly their input) to the owners of the ML system.

Associated controls. Note that the labels refer to the original risks (above) which have controls that may help alleviate some of the risk directly:

[inference:1:online] An ML system in production can be refreshed to a known state, reset, or otherwise "cleaned" periodically. This can limit the window for online attack.

[inference:3:hosting] Take care to isolate engineering ML systems from production systems. Production systems in particular should be properly hardened and monitored.

---

9. **Output risks:** Keep in mind that the entire purpose of creating, training, and evaluating a model may be so that its output serves a useful purpose in the world. The second most obvious direct attack against an ML system will be to attack its output.

   [output:1:direct] An attacker tweaks the output stream directly. This will impact the larger system in which the ML subsystem is encompassed. There are many ways to do this kind of thing. Probably the most common attack would be to interpose between the output stream and the receiver. Because models are sometimes opaque, unverified output may simply be used with little scrutiny, meaning that an interposing attacker may have an easy time hiding in plain sight.

   [output:2:provenance] ML systems must be trustworthy to be put into use. Even a temporary or partial attack against output can cause trustworthiness to plummet.

   [output:3:miscategorization] Adversarial examples (see [input:1:adversarial examples]) lead to fallacious output. If those output escape into the world undetected, bad things can happen.

   [output:4:inscrutability] In far too many cases with ML, nobody is really sure how the trained systems do what they do. This is a direct affront on trustworthiness and can lead to challenges in some domains such as diagnostic medicine.

   [output:5:transparency] Decisions that are simply presented to the world with no explanation are not transparent. Attacking opaque systems is much easier than attacking transparent systems, since it is harder to discern when something is going wrong.

   [output:6:eroding trust] Causing an ML system to misbehave can erode trust in the entire discipline. A GAN that produces uncomfortable sounds or images provides one example of how this might unfold.[17:shane]

   [output:7:looped output] See [input:4:looped input]. If system output feeds back into the real world there is some risk that it may find its way back into input causing a feedback loop.

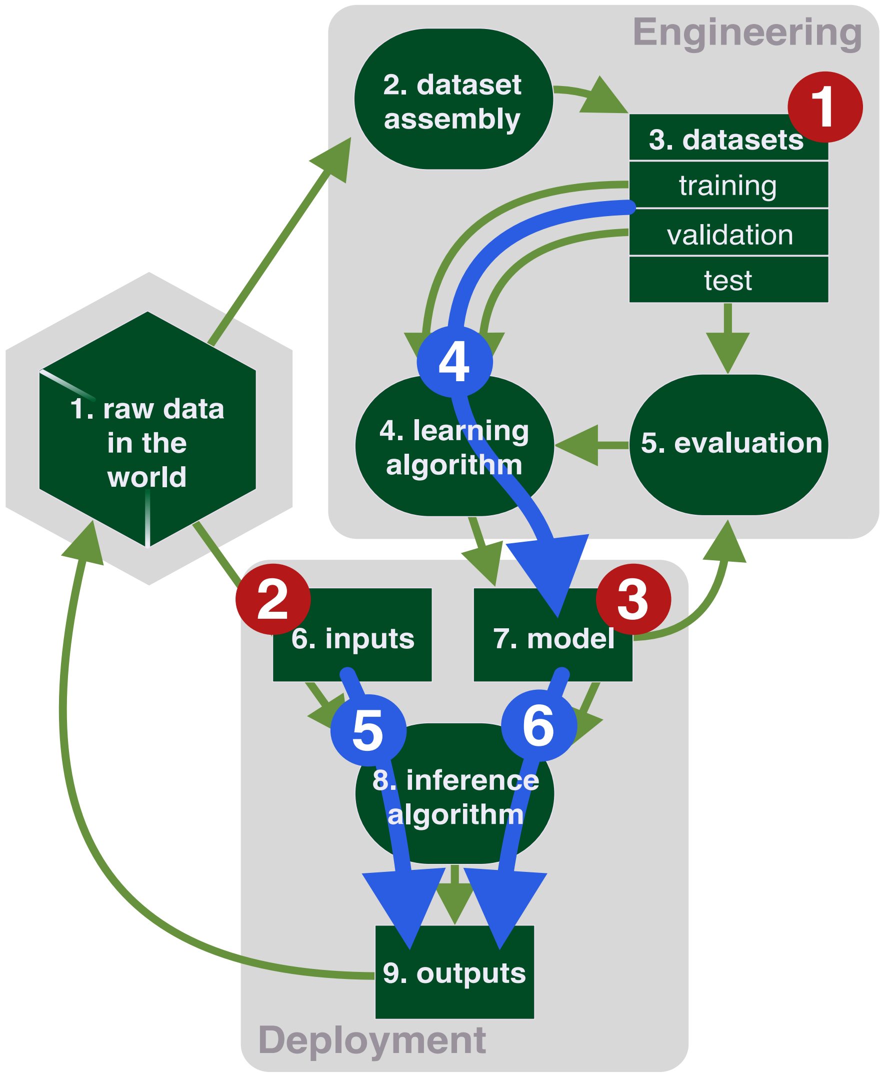

Figure 2: Known attacks and attack surfaces on ML systems. Manipulation attacks are pictured in red at the site of attack: (1) data manipulation. (2) input manipulation. (3) model manipulation. Extraction attacks are pictured in blue, showing the flow of information: (4) data extraction. (5) input extraction. (6) model extraction. Attack surfaces roughly correspond to gray plates: deployment, engineering, and data sources.

Mapping Known Attacks to our Model

In this section we briefly consider direct attacks on ML algorithms. See Figure 2. These attacks are closely related to the security risks we have enumerated above, but they are not the same. Attacks are distinct in that they may leverage multiple risks. You can think of a specific attack as a coordinated exploit of a set of risks that results in system compromise. For the most part we will ignore attacks on ML infrastructure or attacks that specifically circumvent ML-based defense.

We classify attacks on ML systems based on how and where (and to some degree when) the system is compromised. An attack may manipulate the behavior (attacking operational integrity) or extract information (attacking confidentiality). Additionally attacks can affect the training data, run-time inputs, and the model used for inference. Attacks can disrupt both the engineering stages of developing an ML system as well as a deployed ML system. The two axes of how and where a system is compromised lead to a taxonomy with six categories:

(1) **Data manipulation**, also called a "poisoning"[18:kloft] or "causative" attack is a manipulation attack via the training process.[5:barreno] An attacker modifies a data corpus used to train an ML system in order to impair or influence the system's behavior. For example, an attacker may publish bogus data to influence financial time-series forecasting models[19:alfeld] or interfere with medical diagnoses.[20:mozzaffari-kermani]

(2) **Input manipulation**, including by "adversarial examples," is a manipulation attack on an ML model at inference time (or test time).[14:goodfellow] In this case, an attacker concocts an input to an operating ML system which reliably produces a different output than intended. Examples include a stop sign being classified as a speed limit sign;[21:eykholt] a spam email being classified as not spam;[22:biggio] or a vocal utterance being transcribed as an unrelated text.[23:carlini] (For a survey of input manipulation techniques on deep learning and classical ML systems see [Yuan19][16:yuan] and [Biggio13][22:biggio], respectively.) Note that in the online setting, runtime inputs and training data may not be distinct. However, we can say that input manipulation compromises behavior toward the malicious input, while data manipulation compromises behavior toward future inputs—the methods of attack and the security implications are distinct.

(3) **Model manipulation**, also called "backdooring"[24:gu] or a "supply chain" attack occurs when an attacker publishes a model with certain latent behavior, to be unwittingly adopted by third parties and later exploited by the attacker.[25:kumar] It is common in the deep learning community to release models under a permissive open source license; given the prevalence of code reuse and transfer learning we believe this potential attack and defenses against it deserves greater scrutiny.

(4) **Data Extraction**, commonly called "inference attacks" (including membership inference, attribute inference, and property inference) and also sometimes called "model inversion," is when an attacker extracts details of the data corpus an ML model was trained on by querying the model or inspecting it directly.[26:ateniese] Research in deep learning often focuses on the model to the exclusion of the data, yet data are known to be crucially important to a trained system's behavior. Though research is often conducted on public datasets, real-world ML systems involve proprietary data with serious privacy implications.

(5) **Input extraction**, sometimes also called "model inversion" applies in cases where model outputs are public but inputs are secret; an attacker attempts to recover inputs from outputs.[27:fredrikson] For example, inferring features of medical records from the dosage recommended by an ML model,[27:fredrikson] or producing a recognizable image of a face given only the identity (classification in a face-recognition model) and confidence score.[28:wang]

(6) **Model extraction** occurs when an attacker targets a less-than-fully-white-box ML system, attempting to "open the box" and copy its behavior or parameters. Model extraction may function as theft of a proprietary model or may enable white-box attacks on a formerly black-box model.[29:papernot,30:gilmer]

Complementing and contrasting with our taxonomy, NIST recently published a draft taxonomy of adversarial machine learning (AML) which aims to create a taxonomy not just of attacks, but also defenses and consequences.[31:tabassi] At the top level they consider three categories of attack target (physical, digital representation, or ML approach), attack techniques, and knowledge that the attacker may have of the system being attacked.

Overall, the NIST taxonomy and terminology glossary are helpful navigational tools for current literature on AML. Our taxonomy of ML attacks is more focused on practitioners trying to secure an ML system. As such, it is directly grounded on a simpler model for an ML system and more directly describes established categories of system compromise. This makes our taxonomy much simpler, with a clear focus on where and how the system is attacked while still taking into consideration other basic categories in the NIST draft. For example, we find that the our dimension of a how a system is compromised maps effectively to the NIST draft sense of consequence. We also find that our six category approach is straightforward to specialize to specific ML approaches and systems as covered in the targets branch of the NIST taxonomy. Thus we can easily apply our approach of where and how to the various modalities of ML such as supervised, unsupervised, and reinforcement learning.

Attack Surface Estimation

In security, an attack surface us defined as the sum of different locations in a system where an attacker can try to manipulate input, directly impact system processing, or extract data. Keeping a system's attack surface as small as possible is a basic security measure.

Practically, we identify three main attack surfaces for ML systems: deployment, engineering, and data sources. See Figure 2.

Deployment is the most straightforward surface to attack, comprising the inference software and model itself. A deployed ML system includes supporting hardware and software (e.g. web servers); an attacker can typically study an API or hardware device at length to develop an attack. Roughly, the bottom plate in our diagram corresponds to deployment, but note that this distinction is less appropriate in the online learning regime. Conventional computer security is important here, as is understanding how information leaks between model, input, and output.

The ML engineering process is more remote from the system's behavior yet fully determines it. Sensitive data may be most exposed during the engineering process. Roughly, this is the upper plate in Figure 2, though note that the inference algorithm and model are what move from engineering to deployment. Operational security is important here.

Data sources are still more remote from an ML system's behavior but may be particularly easy for an attacker to manipulate undetected. Attacks on data sources should be anticipated when collecting and assembling datasets.

System-Wide Risks and Broad Concerns

To this point, our coverage of ML security risks has been confined to a component-based view. In addition to risks grounded in components, there are a number of system-wide risks that emerge only at the system level or between and across multiple components. We identify and discuss system-wide risks here:

[system:1:black box discrimination]
Many data-related component risks lead to bias in the behavior of an ML system. ML systems that operate on personal data or feed into high impact decision processes (such as credit scoring, employment, and medical diagnosis decisions) pose a great deal of risk. When biases are aligned with gender, race, or age attributes, operating the system may result in discrimination with respect to one of these protected classes. Using biased ML subsystems is is definitely illegal in some contexts, may be unethical, and is always irresponsible.

[system:2:overconfidence]
When an ML system with a particular error behavior is integrated into a larger system and its output is treated as high confidence data, users of the system may become overconfident in the operation of the system for its intended purpose. A low scrutiny stance with respect to the overall system makes it less likely that an attack against the ML subsystem will be detected. Developing overconfidence in ML is made easier by the fact that ML systems are often poorly understood and vaguely described. (See [output:5:transparency].)

[system:3:loss of confidence]
Any ML system can and will make mistakes. For example, there are limitations to how effective the prediction of a target variable or class can be given certain input. If system users are unaware of the subtleties of ML, they may not be able to account for "incorrect" behavior. Lost confidence may follow logically. Ultimately, users may erroneously conclude that the ML system is not beneficial to operation *at all* and thus should be disregarded. In fact the ML system may operate on average much more effectively than other classifying technology and may be capable of scaling a decision process beyond human capability. Throwing out the baby with the bathwater is an ML risk. As an example, consider what happens when self-driving cars kill pedestrians.

[system:4:public perception]
Confidence related risks such as [system:2:overconfidence] and [system:3:loss of confidence] are focused on the impact that common ML misunderstandings have on users of a system. Note that such risks can find their way out into society at large with impacts on policy-making (regarding the adoption or role of ML technologies) and the reputation of a company (regarding nefarious intentions, illegality, or competence). A good example is the Microsoft chatbot, Tay, which learned to converse by parsing raw twitter content and ultimately exhibited racist, xenophobic, and sexist behavior as a result. Microsoft pulled the plug on Tay. Tay was a black eye for ML in the eyes of the public.[32:jagielski]

[system:5:error propagation]
When ML output becomes input to a larger decision process, errors arising in the ML subsystem may propagate in unforeseen ways. For example, a classification decision may end up being treated as imputed metadata or otherwise silently impact a conditional decision process. The evaluation of ML subsystem performance in isolation from larger system context may not take into account the "regret" this may incur. That is, methods that evaluate ML accuracy may not evaluate utility, leading to what has been called regret in the ML literature.

[system:6:cry wolf]
When an ML subsystem operating within a larger system generates too many alarms, the subsystem may be ignored. This is particularly problematic when ML is being applied to solve a security problem like intrusion or misuse detection. False alarms may discourage users from paying attention, rendering the system useless.

[system:7:data integrity]
If ML system components are distributed, especially across the Internet, preserving data integrity between components is particularly important. An attacker in the middle who can tamper with data streams coming and going from a remote ML component can cause real trouble.

[system:8:insider]
As always in security, a malicious insider in an ML system can wreak havoc. Note for the record that data poisoning attacks (especially those that subtly bias a training set) can already be hard to spot. A malicious insider who wishes not to get caught would do well to hide in the data poisoning weeds.

[system:9:API encoding]
Data may be incorrectly encoded in a command, or vice versa. When data and API information are mixed, bad things happen in security. Know that APIs are a common attack target in security and are in some sense your public front door. How do you handle time and state? What about authentication?

[system:10:denial of service]
Denial of service attacks have broad impact when service access impacts a decision process. When an ML system fails, recovery may not be possible. If you decide to rely entirely on an ML system that fails, recovery may not be possible, even if all of the data that feed the ML system are still around.

BIML

PART TWO: ML Security Principles

Security Principles and Machine Learning

In security engineering it is not practical to protect against every type of possible attack. Security engineering is an exercise in risk management. One approach that works very well is to make use of a set of guiding principles when designing and building systems. Good guiding principles tend to improve the security outlook even in the face of unknown future attacks. This strategy helps to alleviate the "attack-of-the-day" problem so common in early days of software security (and also sadly common in early approaches to ML security).

In this section we present ten principles for ML security lifted directly from *Building Secure Software* and adapted for ML.[1:viega] The goal of these principles is to identify and to highlight the most important objectives you should keep in mind when designing and building a secure ML system. Following these principles should help you avoid lots of common security problems. Of course, this set of principles will not be able to cover every possible new flaw lurking in the future.

Some caveats are in order. No list of principles like the one presented here is ever perfect. There is no guarantee that if you follow these principles your ML system will be secure. Not only do our principles present an incomplete picture, but they also sometimes conflict with each other. As with any complex set of principles, there are often subtle tradeoffs involved.

Clearly, application of these ten principles must be sensitive to context. A mature risk management approach to ML provides the sort of data required to apply these principles intelligently.

Principle 1: Secure the Weakest Link

Principle 2: Practice Defense in Depth

Principle 3: Fail Securely

Principle 4: Follow the Principle of Least Privilege

Principle 5: Compartmentalize

Principle 6: Keep It Simple

Principle 7: Promote Privacy

Principle 8: Remember That Hiding Secrets Is Hard

Principle 9: Be Reluctant to Trust

Principle 10: Use Your Community Resources

What follows is a treatment of each of the ten principles from an ML systems engineering perspective.

Principle 1: Secure the Weakest Link

Security people are quick to point out that security is like a chain. And just as a chain is only as strong as the weakest link, an ML system is only as secure as its weakest component. Want to anticipate where bad guys will attack your ML system? Well, think through which part would be easiest to attack and what the attacker's goals might be. What really matters is the easiest way for the attacker to achieve those goals. For a first stab at attack surface analysis, see Figure 2 and the associated text above.

ML systems are different from many other artifacts that we engineer because the data in ML are just as important (or sometimes even more important) as the learning mechanism itself. That means we need to pay much more attention to the data used to train, test, and operate an ML system than we might in a standard system.

In some sense, this turns the idea of an attack surface on its head. To understand what we mean, consider that the training data in an ML system may often come from a public location—that is, one that may be subject to poor data protection controls. If that's the case, perhaps the easiest way to attack an ML system of this flavor would be through polluting or otherwise manipulating the data before they even arrive. An attacker wins if they get to the ML-critical data before the ML system even starts to learn. Who cares about the public API of the trained up and operating ML system if the data used to build it were already maliciously constructed?

Thinking about ML data as money makes a good exercise. Where does the "money" (that is, data) in the system come from? How is it stored? Can counterfeit money help in an attack? Does all of the money get compressed into high value storage in one place (say the weights and thresholds learned in the ML systems' distributed representation)? How does money come out of an ML system? Can money be transferred to an attacker? How would that work?

Let's stretch this analogy even farther. When it comes to actual money, a sort of perverse logic pervades the physical security world. There's generally more money in a bank than a convenience store, but which one is more likely to be held up? The convenience store, because banks tend to have much stronger security precautions. Convenience stores are a much easier target. Of course the payoff for successfully robbing a convenience store is much lower than knocking off a bank, but it is probably a lot easier to get away from the convenience store crime scene. In terms of our analogy, you want to look for and better defend the convenience stores in your ML system.

ML has another weird factor that is worth considering—that is that much of the source code is open source and re-used all over the place. Should you trust that algorithm that you snagged from github? How does it work? Does it protect those oh so valuable data sets you built up? What if the algorithm itself is sneakily compromised? These are some potential weak links that may not be considered in a traditional network security stance.

Identifying the weakest component of a system falls directly out of a good risk analysis. Given good risk analysis information, addressing the most serious risk first, instead of a risk that may be easiest to mitigate, is always prudent. Security resources should be doled out according to risk. Deal with one or two major problems, and move on to the remaining ones in order of severity. You can make use of the ML security risks we identify in this document as a starting point for an in-depth analysis of your own system.

Of course, this strategy can be applied forever, because 100% security is never attainable. There is a clear need for some stopping point. It is okay to stop addressing risks when all components appear to be within the threshold of acceptable risk. The notion of acceptability depends on the business proposition.

All of our analogies aside, good security practice dictates an approach that identifies and strengthens weak links until an acceptable level of risk is achieved.

Principle 2: Practice Defense in Depth

The idea behind defense in depth is to manage risk with diverse defensive strategies (sometimes called controls), so that if one layer of defense turns out to be inadequate, another layer of defense hopefully prevents a full breach.

Let's go back to our example of bank security. Why is the typical bank more secure than the typical convenience store? Because there are many redundant security measures protecting the bank, and the more measures there are, the more secure the place is.

Security cameras alone are a deterrent for some. But if people don't care about the cameras, then a security guard is there to defend the bank physically with a gun. Two security guards provide even more protection. But if both security guards get shot by masked bandits, then at least there's still a wall of bulletproof glass and electronically locked doors to protect the tellers from the robbers. Of course if the robbers happen to kick in the doors, or guess the code for the door, at least they can only get at the teller registers, because the bank has a vault protecting the really valuable stuff. Hopefully, the vault is protected by several locks and cannot be opened without two individuals who are rarely at the bank at the same time. And as for the teller registers, they can be protected by having dye-emitting bills stored at the bottom, for distribution during a robbery.

Of course, having all these security measures does not ensure that the bank is never successfully robbed. Bank robberies do happen, even at banks with this much security. Nonetheless, it's pretty obvious that the sum total of all these defenses results in a far more effective security system than any one defense alone.

The defense-in-depth principle may seem somewhat contradictory to the "secure-the-weakest-link" principle because we are essentially saying that defenses taken as a whole can be stronger than the weakest link. However, there is no contradiction. The principle "secure the weakest link" applies when components have security functionality that does not overlap. But when it comes to redundant security measures, it is indeed possible that the sum protection offered is far greater than the protection offered by any single component.

ML systems are constructed out of numerous components. And, as we pointed out multiple times above, the data are often the most important thing from a security perspective. This means that bad actors have as many opportunities to exploit an ML system as there are components, and then some. Each and every component comes with a set of risks, and each and every one of them needs to address those risks head on. But wait, there's more. Defense in depth teaches that vulnerabilities not addressed by one component should be caught by another. In some cases a risk may be controlled "upstream" and in others "downstream."

Let's think about how defense in depth impacts the goal of securing training data in an ML system. A straightforward security approach will attempt to secure sensitive training data behind some kind authentication and authorization system, only allowing the model access to the data while it is actually training. This may well be a reasonable and well-justified practice, but it is by no means sufficient to ensure that no sensitive information in the training data can be leaked through malicious misuse/abuse of the system as a whole. Here's why. Through the training process itself, the training data come to be represented in the model itself.[33:Fredrikson] That means getting to sensitive data *through* the model is a risk. Some ML models are vulnerable to leaking sensitive information via carefully selected queries made to the operating model itself. In other cases, lots of know-how in "learned" form may be leaked through a transfer attack. A second line of defense against these kind of "through the model" attacks against training data might be to anonymize the dataset so that particularly sensitive aspects of the data are not exposed even through the model.

Maintaining a history of queries made by users, and preventing subsequent queries that together could be used to divine sensitive information can serve as an additional defensive layer that protects against these kinds of attack.

Practicing defense in depth naturally involves applying the principle of least privilege to users and operations engineers of an ML system. Identifying and preventing security exploits is much easier when every component limits its access to only those resources it actually requires. In this case, identifying and separating components in a design can help, because components become natural trust boundaries where controls can be put in place and policies enforced.

Defense in depth is especially powerful when each component works in concert with the others.

Principle 3: Fail Securely

Even under ideal conditions, complex systems are bound to fail eventually. Failure is an unavoidable state that should always be planned for. From a security perspective, failure itself isn't the problem so much as the tendency for many systems to exhibit insecure behavior when they fail.

ML systems are particularly complicated (what with all that dependence on data) and are prone to fail in new and spectacular ways. Consider a system that is meant to classify its input. In a very straightforward way, failure in a classifier would constitute giving the wrong answer (*e.g.*, incorrectly reporting that a cat is a tank). What should an ML system do? Maybe it should emit no answer if confidence is low. Or maybe it can flag inaccurate or iffy classifications like this, through say emitting a confidence score. Reporting a confidence score seems like not such a bad thing to do from an engineering perspective. But in some cases, simply reporting what an ML system got wrong or was underconfident about can lead to security vulnerability. As it turns out, attackers can exploit misclassification to create adversarial examples,[30:gilmer] or use a collection of errors en masse to ferret out confidential information used to train the model.[7:shokri] In general, ML systems would do well to avoid transmitting low-confidence classification results to untrusted users in order to defend against these attacks, but of course that seriously constrains the usual engineering approach. This is a case in which failing securely is much more subtle than it may seem at first blush.

Classification results should only be provided when the system is confident that they are correct. In the case of either a failure or a low confidence result, care must be taken that any feedback from the model to a malicious user can't be exploited. Note that many ML models are capable of providing confidence levels along with their other output to address some of these risks. That certainly helps when it comes to understanding the classifier itself, but it doesn't really address information exploit or leakage (both of which are more challenging problems). ML system engineers should carefully consider the sensitivity of their systems' predictions and take into account the amount of trust they afford the user when deciding what to report.

If your ML system has to fail, make sure that it fails securely.

Principle 4: Follow the Principle of Least Privilege

The principle of least privilege states that only the minimum access necessary to perform an operation should be granted, and that access should be granted only for the minimum amount of time necessary.[3:saltzer]

When you give out access to parts of a system, there is always some risk that the privileges associated with that access will be abused. For example, let's say you are to go on vacation and you give a friend the key to your home, just to feed pets, collect mail, and so forth. Although you may trust the friend, there is always the possibility that there will be a party in your house without your consent, or that something else will happen that you don't like. Regardless of whether you trust your friend, there's really no need to put yourself at risk by giving more access than necessary. For example, if you don't have pets, but only need a friend to pick up the mail on occasion, you should relinquish only the mailbox key. Although your friend may find a good way to abuse that privilege, at least you don't have to worry about the possibility of additional abuse. If you give out the house key unnecessarily, all that changes.

Similarly, if you do get a house sitter while you're on vacation, you aren't likely to let that person keep your keys when you're not on vacation. If you do, you're setting yourself up for additional risk. Whenever a key to your house is out of your control, there's a risk of that key getting duplicated. If there's a key outside your control, and you're not home, then there's the risk that the key is being used to enter your house. Any length of time that someone has your key and is not being supervised by you constitutes a window of time in which you are vulnerable to an attack. You want to keep such windows of vulnerability as short as possible—to minimize your risks.

In an ML system, we most likely want to control access around lifecycle phases. In the training phase, the system may have access to lots of possibly sensitive training data. Assuming an offline model (where training is not continuous), after the training phase is complete, the system should no longer require access to those data. (As we discussed when we were talking defense in depth, system engineers need to understand that in some sense all of the confidential data are now represented in the trained-up ML system and may be subject to ML-specific attacks.)

Thinking about access control in ML is useful and can be applied through the lens of the principle of least privilege, particularly between lifecycle phases and system components. Users of an ML system are not likely to need access to training data and test data, so don't give it to them. In fact, users may only require black box API access to a running system. If that's the case, then provide only what is necessary in order to preserve security.

Less is more when it comes to the principle of least privilege. Limit data exposure to those components that require it and then grant access for as short a time period as possible.

Principle 5: Compartmentalize

The risk analysis of a generic ML system we provide in this document uses a set of nine "components" to help categorize and explain risks found in various logical pieces (see Figure 1). Components can be either processes or collections. Just as understanding a system is easier when a system is divided up into pieces, controlling security risk is easier when the pieces themselves are each secured separately. Another way of thinking about this is to compare old fashioned "monolithic" software design to "micro-services" design. In general, both understanding and securing a monolith is much harder than securing a set of services (of course things get tricky when services interact in time, but we'll ignore that for now). In the end we want to eradicate the monolith and use compartmentalization as our friend.

Let's imagine one security principle and see how compartmentalization can help us think it through. Part of the challenge of applying the principle of least privilege in practice (described above) has to do with component size and scope. When building blocks are logically separated and structured, applying the principle of least privilege to each component is much more straightforward than it would be otherwise. Smaller components should by and large require less privilege than the complete system. Does this component involve pre-processed training data that will directly impact system learning? Hmm, better secure those data!

The basic idea behind compartmentalization is to minimize the amount of damage that can be done to a system by breaking up the system into a number of units and isolating processes or data that carry security privilege. This same principle explains why submarines are built with many different chambers, each separately sealed. If a breach in the hull causes one chamber to fill with water, the other chambers are not affected. The rest of the ship can keep its integrity, and people can survive by making their way to parts of the submarine that are not flooded. Unfortunately, this design doesn't always work, as the *Kursk* disaster of the year 2000 showed.

Some ML systems make use of declarative pipelines as an organizational metaphor. Keep in mind that logical pipeline boundaries often make poor trust boundaries when considered from a security perspective. Though logical boundaries are very helpful from an engineering perspective, if you want to create a trust boundary that must be done as an explicit and separate exercise.

Likewise, note that containers are not always the same thing as conceptual components of the sort we have identified in this work. When you are working on compartmentalization, separation at the logical and data level is what you should be after. In many container models used commonly for ML, everything ends up in one large container without internal trust boundaries. Compartmentalization for security requires more separation of concerns.

Another challenge with security and compartmentalization comes when it is time to consider the system as a whole. As we've seen in our generic ML system here, data flow between components, and sometimes those data are security sensitive. When implementing an ML system, considering component risks is a good start, but don't forget to think through the risks of the system as a whole. Harkening back to the principle of least privilege, don't forget to apply the same sort of thinking to the system as a whole after you have completed working on the components.

Principle 6: Keep It Simple

Keep It Simple, Stupid (often spelled out KISS) is good advice when it comes to security. Complex software (including most ML software) is at much greater risk of being inadequately implemented or poorly designed than simple software is, causing serious security challenges. Keeping software simple is necessary to avoid problems related to efficiency, maintainability, and of course, security.

Machine Learning seems to defy KISS by its very nature. ML models involve complicated mathematics that is often poorly understood by implementers. ML frequently relies on huge amounts of data that can't possibly be fully understood and vetted by system engineers. As a result, many ML systems are vulnerable to numerous attacks arising from complexity. It is important for implementers of ML systems to recognize the drawbacks of using complicated classes of ML algorithms and to build security controls around them. Adding controls to an already complicated system may seem to run counter to our simplicity goal, but sometimes security demands more. Striking a balance between achieving defense-in-depth and simplicity, for example, is a tricky task.

KISS should help inform ML algorithm selection as well as ensemble versus simple algorithm selection. What makes an adequate approach varies according to the goals and requirements of the system, yet there are often multiple choices. When such a choice needs to be made, it is important to consider not only the accuracy claims made by designers of the algorithm, but also how well the algorithm itself is understood by engineers and the broader research community. If the engineers developing the ML system don't really deeply understand the underlying algorithm they are using, they are more likely to miss security problems that arise during operations. This doesn't necessarily mean that the latest and greatest algorithms can't be used, but rather that engineers need to be cognizant of the amount of time and effort it takes to understand and then build upon every complex system.

Principle 7: Promote Privacy

Privacy is tricky even when ML is not involved. ML makes things even trickier by in some sense re-representing sensitive and/or confidential data inside of the machine. This makes the original data "invisible" (at least to some users), but remember that the data are still in some sense "in there somewhere." So, for example, if you train up a classifier on sensitive medical data and you don't consider what will happen when an attacker tries to get those data back out through a set of sophisticated queries, you may be putting patients at risk.

When it comes to sensitive data, one promising approach in privacy-preserving ML is differential privacy.[34:abadi] The idea behind differential privacy is to set up privacy restrictions that, for example, guarantee that an individual patient's private medical data never has too much influence on a dataset or on a trained ML system. The idea is to "hide in plain sight" with a goal of ensuring that anything that can be learned about an individual from the released information, can also be learned without that individual's data being included. An algorithm is differentially private if an observer examining the output is not able to determine whether a specific individual's information was used in the computation. Differential privacy can be achieved through the use of random noise that is generated according to a chosen distribution and is used to perturb a true answer. Somewhat counterintuitively, because of its use of noise, differential privacy can also be used to combat overfitting in some ML situations. Differential privacy is a reasonably promising line of research that can in some cases provide for privacy protection.

Privacy also applies to the behavior of a trained-up ML system in operation. We've discussed the tradeoffs associated with providing (or not providing) confidence scores. Sometimes that's a great idea, and sometimes it's not. Figuring out the impact on system security that providing confidence scores will have is another decision that should be explicitly considered and documented.

In short, you will do well to spend some cycles thinking about privacy in your ML system. If you are doing ML on sensitive data, you *must* take privacy risks seriously, and know that there are no magic solutions. (That is, if you are training a model on sensitive data to do something useful, that model must by its very nature reveal something about its training data.)

Principle 8: Remember That Hiding Secrets Is Hard

Security is often about keeping secrets. Users don't want their personal data leaked. Keys must be kept secret to avoid eavesdropping and tampering. Top-secret algorithms need to be protected from competitors. These kinds of requirements are almost always high on the list, but turn out to be far more difficult to meet than the average user may suspect.

ML system engineers may want to keep the intricacies of their system secret, including the algorithm and model used, hyperparameter and configuration values, and other details concerning how the system trains and performs. Maintaining a level of secrecy is a sound strategy for improving the security of the system, but it should not be the only mechanism.

Past research in transfer learning has demonstrated the ability for new ML systems to be trained from existing ones. If transfer learning is known to have been applied, it may facilitate extraction of the proprietary layers trained "on top" of the base model. Even when the base model is not known, distillation attacks allow an attacker to copy the possibly proprietary behavior of a model using only the ability to query the ML system externally. As a result, maintaining the secrecy of the system's design requires more than simply not making the system public knowledge.

A chief concern for ML systems is protecting the confidentiality of training data. Some may attempt to "anonymize" the data used and consider that sufficient. As the government of Australia discovered in 2017, great care must be taken in determining that the data cannot be deanonymized.[35:culnane] Neural networks similarly provide a layer of anonymization by transforming confidential information into weights, but even those weights can be vulnerable to advanced information extraction techniques. It's up to system engineers to identify the risks inherent in their system and design protection mechanisms that minimize security exposure.

Keeping secrets is hard, and it is almost always a source of security risk.

Principle 9: Be Reluctant to Trust

ML systems rely on a number of possibly untrusted, external sources for both their data and their computation. Let's take on data first. Mechanisms used to collect and process data for training and evaluation make an obvious target. Of course, ML engineers need to get their data somehow, and this necessarily invokes the question of trust. How does an ML system know it can trust the data it's being fed? And, more generally, what can the system do to evaluate the collector's trustworthiness? Blindly trusting sources of information would expose the system to security risks and must be avoided.

Next, let's turn to external sources of computation. External tools such as TensorFlow, Kubeflow, and pip can be evaluated based on the security expertise of their engineers, time-proven resilience to attacks, and their own reliance on further external tools, among other metrics. Nonetheless, it would be a mistake to assume that any external tool is infallible. Systems need to extend as little trust as possible, in the spirit of compartmentalization, to minimize the capabilities of threats operating through external tools.

It can help to think of the various components of an ML system as extending trust to one another; dataset assembly could trust the data collectors' organization of the data, or it could build safeguards to ensure normalization. The inference algorithm could trust the model's obfuscation of training data, or it could avoid responding to queries that are designed to extract sensitive information. Sometimes it's more practical to trust certain properties of the data, or various components, but in the interests of secure design only a minimum amount of trust should be afforded. Building more security into each component makes attacks much more difficult to successfully orchestrate.

Principle 10: Use Your Community Resources

Community resources can be a double-edged sword; on the one hand, systems that have faced public scrutiny can benefit from the collective effort to break them. But nefarious individuals aren't interested in publicizing the flaws they identify in open systems, and even large communities of developers have trouble resolving all of the flaws in such systems. Relying on publicly available information can expose your own system to risks, particularly if an attacker is able to identify similarities between your system and public ones.

Transfer learning is a particularly relevant issue to ML systems. While transfer learning has demonstrated success in applying the learned knowledge of an ML system to other problems, knowledge of the base model can sometimes be used to attack the student.[28:wang] In a more general sense, the use of publicly available models and hyperparameters could expose ML systems to particular attacks. How do engineers know that a model they use wasn't deliberately made public for this very purpose? Recall our discussion of "Trojan models" from the attack taxonomy section above.

Public datasets used to train ML algorithms are another important concern. Engineers need to take care to validate the authenticity and quality of any public datasets they use, especially when that data could have been manipulated by unknown parties.

At the core of these concerns is the matter of trust; if the community can be trusted to effectively promote the security of their tools, models, and data, then community resources can be hesitantly used. Otherwise, it would be better to avoid exposing systems to unnecessary risk. After all, security problems in widely-used open-source projects have been known to persist for years, and in some cases decades, before the community finally took notice.

Putting this Risk Analysis to Work

This document presents a basic architectural risk analysis and a set of 78 specific risks associated with a generic ML system. We organize the risks by common component and also include some system-wide risks. These risk analysis results are meant to help ML systems engineers in securing their own particular ML systems.

In our view ML systems engineers can devise and field a more secure ML system by carefully considering the risks in this document while designing, implementing, and fielding their own specific ML system. In security, the devil is in the details, and we attempt to provide as much detail as possible regarding ML security risks and some basic controls.

We have also included a treatment of security principles as adapted in *Building Secure Software* and originally published in 1972 by Saltzer and Shroeder.[1:viega, 3:saltzer] This treatment can help provide an important perspective on security engineering for researchers working in ML.

BIML

Acknowledgements

The authors thank the following people for their insightful comments and suggestions on earlier drafts of this document: Doug Blank, Michelle Casbon, Richard Danzig, David Evans, Melanie Mitchell, and Brian Sletten.

References

See the Berryville Institute of Machine Learning Annotated Bibliography for more commentary and references.

[1] Viega, John, and Gary McGraw, ***Building Secure Software***, Addison-Wesley, 2001.

[2] McGraw, Gary, ***Software Security***, Addison-Wesley, 2006. See chapter 5.

[3] Saltzer, J.H., and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE,* 9(63), 1975.

[4] Papernot, Nicholas, "A Marauder's Map of Security and Privacy in Machine Learning," ***arXiv:1811.01134 [cs]***, Nov. 2018.

[5] Barreno, Marco, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. Doug Tygar. "Can machine learning be secure?" In ***Proceedings of the 2006 ACM Symposium on Information, computer and communications security***, pp. 16-25. ACM, 2006.

[6] Wu, Yonghui, et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." ***arXiv preprint arXiv:1609.08144*** (2016).

[7] Shokri, R., M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in ***Proc. 2017 IEEE Symp. Security Privacy***, 2017, pp. 3–18.

[8] Barreno, M., Blaine Nelson, Anthony D. Joseph, J.D. Tygar, "The Security of Machine Learning." ***Machine Learning, Vol.81, Issue 2, pp 121-148,*** November 2010.

[9] Phillips, P. Jonathon, Fang Jiang, Abhijit Narvekar, Julianne Ayyad, and Alice J. O'Toole. "An other-race effect for face recognition algorithms." ACM Transactions on Applied Perception (TAP) 8, no. 2 (2011): 14.

[10] Sculley, D., Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. "Machine learning: The high interest credit card of technical debt." (2014).

[11] Ribeiro, M.T., S. Singh, and C. Guestrin. "Anchors: High-precision model-agnostic explanations." In *Thirty-Second AAAI Conference on Artificial Intelligence,* April 2018.

[12] Alfeld, S., Zhu, X., Barford, P., "Data Poisoning Attacks against Autoregressive Models." *AAAI Conference on Artificial Intelligence*, North America, Feb. 2016. Available at: <https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12049>. Date accessed: 11 Sep. 2019.

[13] McGraw, Gary, Richie Bonett, Harold Figueroa, and Victor Shepardson. "Securing Engineering for Machine Learning," *IEEE Computer*, Volume 52, Number 8, pages 54-57.

[14] Goodfellow, Ian, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." In *Proc. ICLR*, 2015.

[15] Reiter, E., "A Structured Review of the Validity of BLEU." *Computational Linguistics* 44(3):393-401, September 2018.

[16] Yuan, Xiaoyong, Pan He, Qile Zhu, and Xiaolin Li, "Adversarial Examples: Attacks and Defenses for Deep Learning." *IEEE Transactions on Neural Network Learning Systems*, 2019, pp. 1–20.

[17] Shane, Janelle, *You Look Like a Thing and I Love You*, Voracious (November 5, 2019).

[18] Kloft, Marius, and Pavel Laskov. "A poisoning attack against online anomaly detection." In *NIPS Workshop on Machine Learning in Adversarial Environments for Computer Security*. 2007.

[19] Alfeld, Scott, Xiaojin Zhu, and Paul Barford. "Data poisoning attacks against autoregressive models." In *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[20] Mozaffari-Kermani, Mehran, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K. Jha. "Systematic poisoning attacks on and defenses for machine learning in healthcare." *IEEE journal of biomedical and health informatics*, 19(6):1893-1905, 2014.

[21] Eykholt, Kevin, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. "Robust physical-world attacks on deep learning models." arXiv preprint arXiv:1707.08945 (2017).

[22] Biggio, Battista, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. "Evasion attacks against machine learning at test time." In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 387-402. Springer, Berlin, Heidelberg, 2013.

[23] Carlini, Nicholas, and David Wagner. "Audio adversarial examples: Targeted attacks on speech-to-text." In *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 1-7. IEEE, 2018.

[24] Gu, T., B. Dolan-Gavitt, and S. Garg. "Badnets: Identifying vulnerabilities in the machine learning model supply chain." arXiv preprint arXiv:1708.06733 (2017)

[25] Kumar, R.S.S., D. O Brien, K. Albert, S. Viljöen, J. Snover, "Failure Modes in Machine Learning Systems." arXiv preprint 1911.11034 (2019)

[26] Ateniese, G., G. Felici, L.V. Mancini, A. Spognardi, A. Villani, and D. Vitali. "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers." arXiv preprint arXiv:1306.4447 (2013).

[27] Fredrikson, Matthew, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing." In *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 17-32. 2014.

[28] Wang, B., Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With Great Training Comes Great Vulnerability: Practical Attacks against Transfer Learning," *27th USENIX Security Symposium*, 2018, pp. 1281–1297.

[29] Papernot, Nicolas, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. "Practical black-box attacks against machine learning." In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 506-519. ACM, 2017.

[30] Gilmer, Justin, Ryan P. Adams, Ian Goodfellow, David Andersen, and George E. Dahl. "Motivating the Rules of the Game for Adversarial Example Research." arXiv preprint 1807.06732 (2018)

[31]Tabassi, E., K. Burns, M. Hadjimichael, A. Molina-Markham, J. Sexton, "A Taxonomy and Terminology of Adversarial Machine Learning", NIST Technical Draft, Oct 2019, https://doi.org/10.6028/NIST.IR.8269-draft

[32] Jagielski, M. , A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, B. Li "Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning" arXiv preprint 1804.00308 (2018)

[33] Fredrikson, M., S. Jha, and T. Ristenpart, "Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures," *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.

[34] Abadi, Martin, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang, "Deep Learning with Differential Privacy," In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). ACM, New York, NY, USA, 308-318. DOI: https://doi.org/10.1145/2976749.2978318

[35] Culnane, Chris, Benjamin Rubinstein, Vanessa Teague. "Understanding the Maths is Crucial for Protecting Privacy." Technical Report from Department of Computing and Information Systems, University of Melbourne. (Published Sept 29, 2016; Accessed Oct 28, 2019.)

BIML