MITRE

MTR200598
MITRE TECHNICAL REPORT

Odyssey:

A Systems Approach to Machine Learning Security

Ransom Winder, PhD
Joseph Jubinski
Chris Giannella, PhD
Malachi Jones, PhD

April 2021

The views, opinions and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official government position, policy, or decision, unless designated by other documentation.

Approved for Public Release; Distribution Unlimited. Public Release Case Number 20-3341

©2021 The MITRE Corporation.
All rights reserved.

Annapolis Junction, MD

Odyssey
© 2021, The MITRE Corporation

© 2021, The MITRE Corporation

1 Abstract

Using software exposes vulnerabilities that are susceptible to attacks with serious consequences. While there are many different vulnerabilities, the consequences fall into a small number of categories. This paper explains how consequences of attacks on Machine Learning (ML) vulnerabilities fall into these same categories. These consequences are then aligned to ML-specific attacks, the contexts in which they occur, and the established methods for mitigating them. These defensive countermeasures can support the security of the ML elements of a system and lead to greater assurance that systems using ML will operate as planned. This paper provides a systems approach to addressing attacks, consequences, and mitigations for systems using ML. It explains each of these over the lifecycle of an ML technology, providing clear explanations of what to worry about, when to worry about it, and how to mitigate it while presuming little incoming knowledge of ML specifics.

Scope: consequences and mitigations of attacks on ML in the context of ML systems.

2 Introduction

The Intelligence Community is examining the potential of fully automating well-defined processes and augmenting human expertise with systems that employ Artificial Intelligence and Machine Learning (ML) [1]. ML utilizes data to develop automated decision-making capabilities that, when embedded in larger systems, have the potential to increase the accuracy, efficiency, and capacity of such systems. To instill trust in ML-based systems, their vulnerabilities must be recognized, the attacks that can be perpetrated against them must be anticipated, and the potential damage must be mitigated.

ML provides the power to enable decision-making at scale in a uniform and consistent manner. This begs the question: how can an adversary exploit this power? If an adversary can control or influence decision processes or behaviors of a computing system (e.g., through malware), the consequences are not limited to cyber and can be realized in the physical world (e.g., damage caused by Stuxnet).

Despite its power, ML can be quite brittle and susceptible to being exploited or undermined. For example, Cylance[1] employs ML for malware detection, but security researchers were able to bypass detection by exploiting how Cylance's ML model works. The researchers added additional strings to malware that preserved its intended behavior while allowing it to avoid detection [2]. Modern malware detection systems look for abnormal execution behavior. However, ML offers a new frontier for malware with the goal of indirectly modifying system decision-making, which is substantially more difficult to recognize than abnormal execution behavior.

As another example, in December 2018, Tumblr[2] changed its policy to reduce the prevalence of "adult" content on its platform. Users whose content was flagged received emails from Tumblr and the flagged posts were made private and viewable only by the users [3]. To scale to the size

[1] https://www.cylance.com/content/dam/cylance/pdfs/data_sheets/CylancePROTECT.pdf
[2] https://www.tumblr.com/

of its content, Tumblr relied on ML [4] to flag adult content and people to "…help train and keep our systems in check" [5]. To thwart the automated flaggers, some Tumblr users developed straightforward methods of altering posts with adult content by including innocuous and incongruent content [6].

Even if existing security systems are in place, ML creates opportunities for new and unique attacks. However, the negative consequences of these attacks fall into well-established categories. Examining these provides a good basis for knowing where the greatest concerns lie. Therefore, to explain the picture of ML vulnerabilities, threats, and mitigations, we start by examining these negative consequences, gradually mapping them back to the unique qualities of ML. We aim to provide readers with these insights without requiring prior understanding of ML nuances.

We describe the types of damage that software systems typically face and tie them to recognized categories of consequences that are specific to systems that employ ML. We then explain the vectors of attacks that lead to these consequences. We then describe ML itself in its broadest categories, including the lifecycle from inception to deployment and execution. We then identify where vulnerabilities exist in the lifecycle that allow threats to initiate ML-targeted attacks on the system. We then offer a deeper examination of ML vulnerabilities, attacks, and mitigations through different examples.

Understanding the lifecycle of an ML system, where vulnerabilities lie in the lifecycle, and the damage that can be done if an attack exploits those vulnerabilities allows informed assessments of the risks incurred by employing ML. Our discussion of ML vulnerabilities, attacks, and mitigations utilizes the taxonomy developed in NISTIR 8269 [7]. Our major point of departure lies in the mapping of these concepts to the ML lifecycle we articulate in section 6 and our discussion of a systems approach to ML security.

3 Vulnerabilities and Consequences

We begin with a focus on known categories of consequences of attacks on ML vulnerabilities. This categorization is helpful as it allows a system designer to witness and understand the types of consequences that may occur and recognize which are relevant for a given mission. We begin generally, without an immediate interpretation of these consequences or vulnerabilities in a mission context.

For software in general, the Common Weakness Enumeration[3] (CWE) is a list of common software security weaknesses, developed to serve as a baseline for weakness identification, mitigation, and prevention. It provides a common semantics for describing these weaknesses. A vulnerability is a weakness that a threat can exploit to access an asset. While software weaknesses are well understood, they also feature a wide variety of specific examples. In contrast, the consequences of their exploitation can be categorized simply with intuitive and useful distinctions.[4] Moreover, systems that use ML software suffer the very same well-known consequences.

In this section, we explain how the consequences articulated by the CWE align to ML-specific categories of consequences. This provides the foundation for explaining what attacks lead to these consequences, where they might occur in the ML lifecycle, and what can be done to address them.

3.1 Machine Learning Specific Consequences

The exploration of the cybersecurity aspects of ML is relatively young, and those aspects are still being defined. A recent NIST report [8] explored the topic of adversarial machine learning (AML) and proposed a taxonomy that organized attacks, consequences, and defenses in this context. Their research identified three major categories of AML consequences, characterized as violations of integrity, availability, and confidentiality. Each of these categories either bears directly on the ML model, which is the basis for decision-making defined through ML, or on the overall system through exploiting a vulnerability due to the use of ML.

3.1.1 Integrity

When an attack undermines the inferences produced by an ML model, it constitutes a violation of *integrity* [8]. This aligns to CWE impacts on the ML model itself or consequences felt on the overall system. The impacts related to integrity that directly bear on the ML model include:

- Modify data[5]
- Hide activities[6]

The "modify data" impact is realized when the training input (or a signal that influences the agent's model) is compromised, thus undermining the effectiveness of the model. This impact can typically arise as the result of a *poisoning* attack. The "hide activities" impact is realized when an adversary (or the data an adversary creates) adopts features the ML model does not recognize,

[3] https://cwe.mitre.org/index.html
[4] https://cwe.mitre.org/community/swa/priority.html
[5] The unauthorized change of content
[6] The purposeful masking of behavior that should be available for analysis

thereby leaving the adversary's identity undetected and its behavior judged incorrectly. This impact can typically arise as the result of an *evasion* attack. We describe these attacks further in section 4.

The integrity-related consequences felt by the larger system that employs ML include:

- Denial-of-service: unreliable execution[7]
- Execute unauthorized code or commands[8]
- Gain privileges / assume identity[9]
- Bypass protection mechanisms[10]

In the case of the "denial-of-service: unreliable execution" consequence, this can arise when applying a model that has poor or malicious training input, making it another potential outcome of a poisoning attack. Each of the other cases represents a situation where the ML model is used to protect an asset but has been thwarted by the adversary who had gained access to the asset, making them further potential outcomes of evasion attacks.

3.1.2 Availability

When an attack makes an ML model function at an unacceptable speed or fail to function entirely, it constitutes a violation of *availability* [8]. This most strongly relates to the CWE impact described as "denial-of-service: resource consumption."[11] These impacts are realized when an input adopts features that make the model execution take significantly longer, and these can arise as the result of a *sponge* attack, as described in section 4.

3.1.3 Confidentiality

When an attack involves an adversary accessing, extracting, or inferring usable information about the ML model, it constitutes a violation of *confidentiality*. A subcategory of this of particular interest is violation of *privacy*, where the adversary can discover personal information from the ML model's legitimate training inputs [8]. The violation of confidentiality relates most strongly to what CWE lists as "read data"[12] impacts. These can arise as the result of an *inversion* attack, as described in section 4.

3.2 Summary and Risk

While attacks exploiting ML are relatively new in the scope of cybersecurity, the impacts of these attacks on the vulnerabilities exhibited by ML models are not. These impacts align with what has already been observed in a general software security context. This section has established the impacts that can occur given different attacks, and this awareness of what damage can be caused by specific attacks motivates examining the attacks themselves. The examination of attacks in turn

[7] The hindrance of software from performing its function by making the software unable to operate as expected
[8] The ability to perform activities with the software that normally require privileges withheld
[9] The acquisition of privileges specifically to create a persona that can perform unauthorized activities
[10] The circumstances where privileges and security are circumvented to gain access to protected content or capabilities
[11] The hindrance of software from performing its function by denying or using resources required for it to execute
[12] The unauthorized disclosure of content

serves as the basis for describing where the attacks occur in the lifecycle of ML models and how these attacks might be mitigated.

Understanding these impacts also assists in judging risk, which is a calculated assessment of the vulnerabilities in a system's software and the negative potential of using the software. Because risk is tied closely to assessments that draw from a scenario's context, it is difficult to speak in terms crossing all potential circumstances. To remain more general, we do not discuss risks arising from use of ML, although this work may inform those assessments.

4 Attacks

The growing significance of ML systems and the expected reliance on them makes attacks against them more likely. A recent survey of academic literature for NIST [8] discussed three attack vectors on AI systems, all of which apply specifically to ML and have different methods and goals. Broadly, these vectors include poisoning attacks, evasion attacks, and oracle attacks. The complete NIST taxonomy of these attack techniques is reproduced in Figure 4-1.

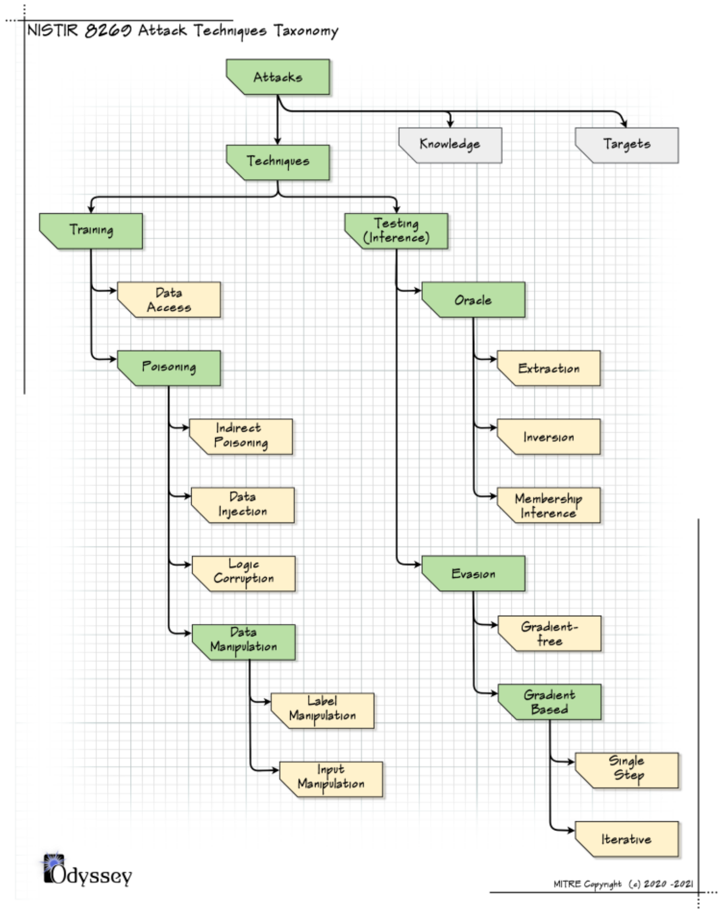

Figure 4-1: NISTIR 8269 Attack Technique Taxonomy

Poisoning attacks occur at the earliest stages of ML. ML systems typically rely on training data and poisoning exploits this by injecting manipulated data into the training process to influence what is learned. This is a representative example of a class of causative attacks, which encompasses attacks that affect ML by changing the training data's distribution [9].

A separate set of attacks shift the emphasis from modifying ML systems to manipulating their perceptions or exposing the sensitive information that drives their decision-making processes. While sometimes broadly characterized as exploratory attacks [9], more specifically they include evasion, impersonation, and oracle (or inversion) attacks.

Evasion attacks involve an adversary creating malicious and misleading inputs after training is complete to cause a misclassification or a misprediction by the system. Related to this are impersonation attacks, in which an attacker creates adversarial samples that will be misclassified when compared to the originals [9]. These classes of attack seek to manipulate or exploit the established behavior of an ML system.

Oracle attacks (or inversion attacks) involve an adversary querying the algorithm with the goal of compromising the confidentiality of the underlying ML system or the data that defines its behavior. While some—although not all—of the implementations of the other attack methods presume knowledge either about the training data or the underlying model and its parameterization, oracle (or inversion) attacks are specifically engineered to target capturing these and exposing any withheld information encoded in the models.

The above attacks fit into the NIST taxonomy, but a few other significant categories exist. There are tampering attacks that actively seek to alter the behavior of the ML system at run-time. There is also a newer type of attack introduced recently that relies on an exploitation that different inputs of the same size can exhibit radically different energy or time consumption. These high-energy/time consumption inputs were called “sponge examples” suggesting a new kind of sponge attack at run-time [10].

These attacks can exploit different vulnerabilities, but they may apply across many different realized ML systems. A more specific survey of instances of these attacks is offered in section 8, but that examination benefits from some elaboration of what makes different classes of ML distinct, which is covered next.

5 Machine Learning

ML methods can largely be broken down into three categories: supervised learning, unsupervised learning, and reinforcement learning [7] as shown in Figure 5-1. As with most categorizations of a large field of endeavor, this one has flaws, namely there exist ML methods that cannot be cleanly assigned to one of these categories, but as an organizing principle, this categorization is helpful and widely used.

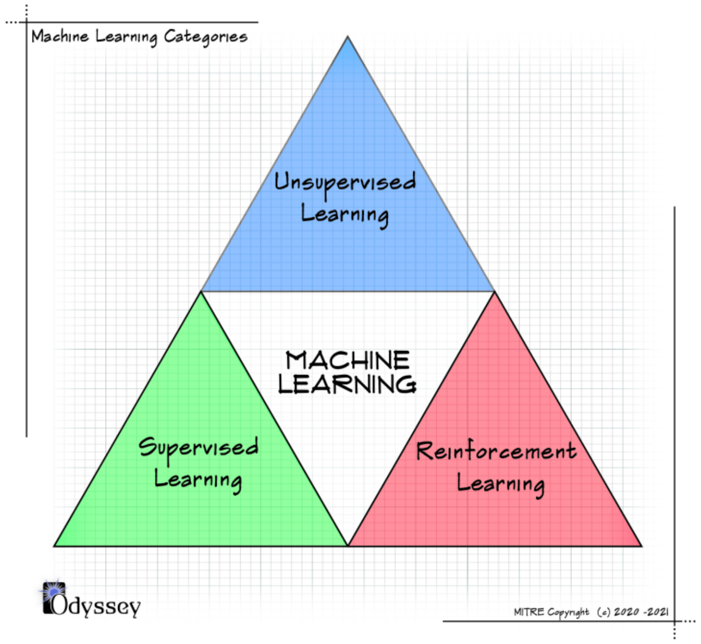

Figure 5-1: Machine Learning Categories

Supervised learning uses instances of data that have been assigned labels as informative examples to accomplish the task of learning a function that maps new instances of data to their labels. This allows for a model to learn by way of examples during training and take what has been learned to recognize through analysis that certain characteristics indicate a specific category or a specific assessment.

In contrast to supervised learning, *unsupervised learning* requires data but does not leverage pre-assigned labels. Instead it discovers patterns that exist in the data set. The examples provided allow the learning methods to produce evaluations of new instances based on the training data's distribution.

Semi-supervised learning is an approach applying supervised and unsupervised techniques in concert to address scenarios where there may not be adequate labeled data to perform supervised learning to yield a model that meets performance (e.g., accuracy in terms of classification) metrics. These scenarios occur often in practice. A common approach leverages a small amount of labeled data to generate labels for unlabeled data through unsupervised ML. The newly generated labels combined with the original labeled data can be fed into a supervised learning algorithm to produce a model that can out-perform models that are derived solely from the original labeled data.

Finally, *reinforcement learning* is not reliant on data and labels, but rather the interaction of an agent, or decision-maker, with its environment. The learning is accomplished through a reward signal that reinforces behaviors. Through repeated rewards (or punishments), the agent develops a policy of how to behave in new interactions with the environment.

The following subsections provide a detailed foundational description of each ML category, but a deep analysis of these subsections is not essential to understanding the following sections.

5.1 Supervised Learning

Approaches in this category address the problem of approximating an unknown function that assigns labels y_i from set L to data objects x_i in set D. More formally, this function[13] $f: D \to L$ is based on examples $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n) \in D \times L$, where $y_i \approx f(x_i)$. If L is finite (typically small), this task is called classification. If L is infinite, this task is called regression. The examples are called labeled training data where the labels (the y's) are commonly assumed to be more difficult to obtain than the data objects (the x's).

As a representative example, suppose data objects are greyscale representations of images captured by a satellite and the labels are manually assigned indicators of objects from a fixed universe (e.g., {airplane, vehicle}), these objects appearing in the images. In this example, D is the set of all possible greyscale representations of images at the resolution of the camera, L is the fixed universe of labels, and the x's and y's are pairs of individual greyscale images and their manually assigned labels. In this case, L is finite and small, making the task classification.

Myriad algorithms have been developed for inferring an approximation function $\hat{f}: D \to L$, called the learned model, from the training data. The accuracy of the learned model is typically gauged using a different set of examples $(x'_1, y'_1), (x'_2, y'_2), \cdots, (x'_m, y'_m) \in D \times L$ called labeled testing data.

A major hurdle faced while applying supervised learning in practice often lies in the difficulty of accurately labeling a sufficiently large number of data objects for use in training and testing.

5.2 Unsupervised Learning

Approaches in the unsupervised learning category address the problem of inferring the structure underlying an unlabeled set of data objects $x_1, x_2, \cdots, x_n$. The problem addressed here is less precisely defined than the problem addressed by supervised learning, leading to a wider variety of methods and greater difficulty evaluating their accuracy.

In classical statistics, one widely studied unsupervised learning problem involves inferring explicit properties of an unknown probability distribution from which each point in the dataset is assumed to have been independently generated. If the functional family, e.g. *Gaussian*, of the underlying probability distribution is assumed known, approaches to address this problem are called *parametric*. If no such assumption is made, approaches are called *non-parametric*. Another widely studied unsupervised learning problem involves the assignment of data points to groups (clusters). The goal of the grouping for points in each cluster is to be more "similar" to each other than to those points in other clusters (e.g., organizing malware binaries into groups based on behavior similarity without knowing in advance the taxonomy of malware families present). Examples of commonly used clustering algorithms include *K-means* and *Hierarchical Agglomerative Clustering*. In the realm of text data (the x's are strings of text), unsupervised learning approaches are used to infer word embeddings: a mapping from words to real vectors such that Euclidean distance approximates semantic similarity.

[13] D denotes the domain of possible data objects and L denotes the domain of possible labels.

Unsupervised learning does not require that the data have target labels with which to train a model and thus, in many circumstances, the acquisition of large quantities of data is relatively easy.

5.3 Reinforcement Learning

Approaches in this category address the problem of learning through continual interaction with the environment. Some argue that the basic setup of reinforcement learning more closely resembles human learning than do supervised or unsupervised learning [11]. The setup involves an agent who takes a sequence of actions, receiving for each a reward value from the environment. The goal of the agent is to maximize the total reward received over the lifetime of the interaction.

The problem can be more formally defined as a *Finite Markov Decision Process* [11]. The sets of all possible environment states, S, all possible actions, A, and all possible reward values, R, are assumed finite. At step t, the agent acquires $s_t \in S$, a representation of the environment's state, and selects an action $a_t \in A$. As a result, at step $t+1$, the agent receives a reward $r_{t+1} \in R$ and acquires a new representation of the environment's state $s_{t+1} \in S$. It is assumed, r_{t+1} and s_{t+1} are independent (statistically) of actions and states prior to step t: $s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \cdots$ (i.e. a *Markov* assumption is made).

The agent selects action a_t based on a policy π_t, a collection of conditional probability distributions: Pr $[.|s]$ for all $s \in S$ where the distribution assigns a probability to each action $a \in A$. The agent is free to modify the policy at each step, i.e. π_{t+1} could be different than π_t. The goal of the agent is to continually adjust the policy to maximize the aggregate reward over the entire interaction. This interaction process might be repeated over multiple episodes with the final policy from a previous episode informing the starting policy of the next episode.

A key distinction between supervised and reinforcement learning is that supervised learning has ground truth (i.e. labeled training data set), which informs the classification decision policy (i.e. a mapping of data point to a label). In contrast, reinforcement learning builds beliefs (e.g., optimal vs. suboptimal) about decision policies (i.e. a mapping of history to actions) based on observations within the environment. How an initial policy is selected can vary based on implementation. As an example, Monte Carlo control is an optimization procedure for improving a policy where a random policy is chosen to be an initial policy versus techniques that leverage pre-training with neural networks.

6 Machine Learning Lifecycle

Our background has provided an overview of the common patterns of ML such as supervised, unsupervised, and reinforcement learning. We next explore this from a systems perspective, looking at the general lifecycle of creating, deploying, and maintaining a system that emphasizes leveraging ML models. There are common activities that have led to proposed models of the ML lifecycle [12], and we articulate our own view. We first depict this at a high level that encompasses most typical patterns and then describe how the key components of the system interact over the lifecycle. The simplest representation of the lifecycle's main steps is shown in Figure 6-1, which includes steps for data[14] activities, learning activities, staging learned models in the final system, and runtime where the system is deployed, which can inform how the next iteration of producing a model proceeds. With that established, we describe where differences emerge for systems employing different broad categories of ML.

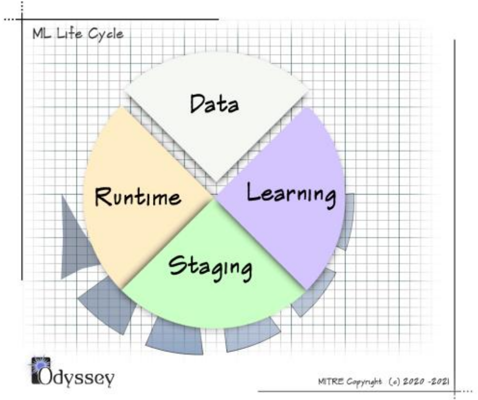

Figure 6-1: Basic Machine Learning Lifecycle

6.1 Machine Learning Stack

To express a system that employs ML at a high-level, we employ a stack diagram as depicted in Figure 6-2. We derive this analogy from the standard protocol stack or network stack of software implementation of communication protocols. In a standard protocol stack diagram, graphically lower layers represent the low-level interactions and higher layers add capability with less presumption of awareness of those lower-level interactions. In our model, the layers represent a similar relationship. The higher layers assume the existence of and depend on the lower layers, but they do not presume that interactions with the higher layers, either with users or other applications, require knowledge of or interactions with the lower layers. As one descends the stack layers, more is offered, providing more insight into the underlying ML requirements, assumptions, and functions. These layers are also stretched to conform to where they interact, and further breakdowns of the high-level layers expose these interactions.

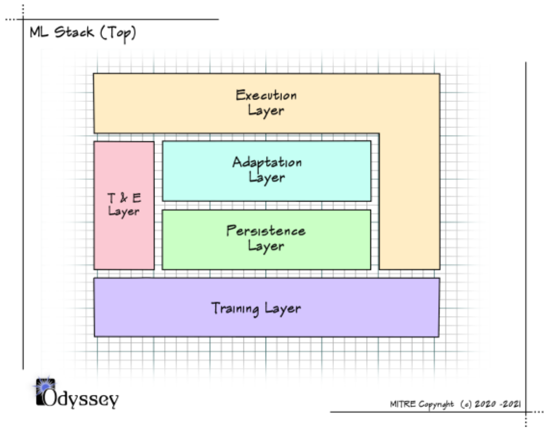

Figure 6-2: Machine Learning Model Stack

Our stack consists of the following layers: Training, T&E (Test & Evaluation), Persistence, Adaptation, and Execution. These divisions were chosen largely based on context and function. The

[14] While common, not every ML method will rely on data. For instance, Reinforcement Learning is principally based on rewarding agent behavior.

primary contextual division we make (Figure 6-3) is between *development*, which encompasses the Training Layer, the T&E Layer, the Persistence Layer, and the Adaptation Layer, and *fielding*, which consists of the Execution Layer. The terms development and fielding are derived from those used by the National Security Commission on Artificial Intelligence (NSCAI) who define development as "designing, building, and testing during development and prior to deployment" and fielding as "deployment, monitoring, and sustainment" [13].

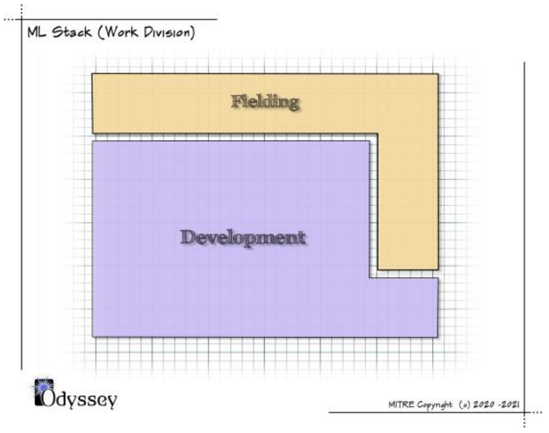

Figure 6-3: ML Model Stack, Work Division

We deem this as the starkest separation in the lifecycle of an ML system because it distinguishes between when a solution is undergoing fabrication and when a solution has been released to operate. The vulnerabilities that occur on either side of that dividing line can have dramatically different impacts, the risks typically being much greater and the impacts worse after deployment. Nevertheless, impacts in development can have consequences felt during fielding.

Further functional divisions of the stack feature major differences between what is true for the three broad categories of ML. We first describe these for supervised and unsupervised learning methods and then for reinforcement learning methods.

6.2 Supervised and Unsupervised Machine Learning Stack

The coarsest divisions by function occur within the development division. The Training Layer is where the ML components of an eventual application are both defined and generated. For most standard variations of ML, this involves triage of the data used to inform the model's creation, selection of methods to make use of the data to create a ML model, and the application of those methods to train a model that will be evaluated, persisted, and adapted for use during execution in a deployed system. The key subdivisions of these activities (categorized as triage, methods, and training) are depicted in Figure 6-4.

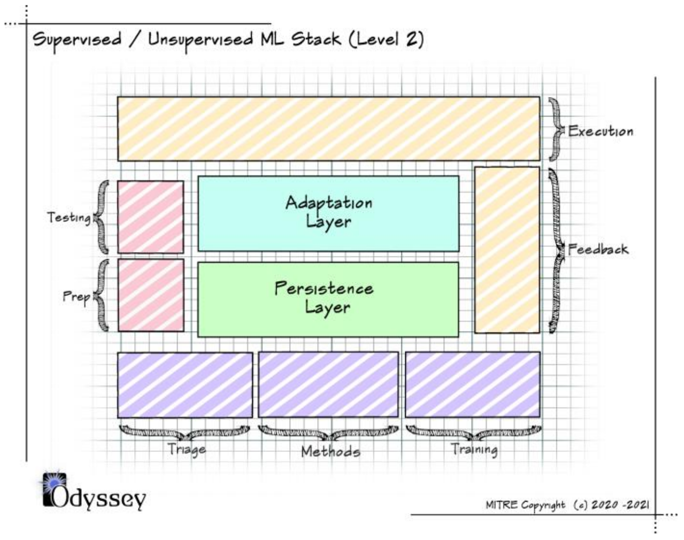

Figure 6-4: Machine Learning Stack, Level 2

The T&E Layer is where components are tested and evaluated. This includes activities preparatory to performing training such as data validation and activities that follow training, such as the evaluation of the models and testing the model in its final system before deployment. This layer's functions depend on having data or trained models, but it does not have to make assumptions about the methods employed to create either. The major subdivisions of these activities (categorized as preparation and testing) are depicted in Figure 6-4.

The Persistence Layer is where models are retained after being trained. This is effectively a store of all models evaluated as worth keeping for use in a deployed system. This layer assumes that training has produced a model, but it does not levy any requirements about how the model is produced (i.e., data used, learning algorithms). However, it should retain the necessary metadata to ensure a model is selected properly downstream for appropriate use in a deployed application.

The Adaptation Layer is where persisted models are readied for deployment in an application. This may be a trivial matter of using a generated solution as-is or may involve changes in representation so that a model retains its ability to make decisions or predictions but is made to conform to the application's expectations of its access or representation.

Constituting the fielding division, the Execution Layer is where an application has been deployed to operate in a real-world environment. It executes, or is executed, and can be monitored for performance to make sure that it is and continues operating as expected. Based on the monitoring, feedback may occur that drives the Training Layer to retrain a new model on new data; this might happen if the performance falls below some acceptable benchmark. The major subdivisions of these activities (categorized as execution and feedback) are depicted in Figure 6-4.

6.3 Supervised and Unsupervised Machine Learning Lifecycle

The stack as we have defined it loosely implies a lifecycle that most typical systems that employ supervised or unsupervised ML follow. As depicted in Figure 6-5, the lifecycle involves several stages that make up sublayers of the stack diagram. While not representative of every supervised and unsupervised ML scenario, like the stack diagram, this is a useful abstraction that applies to most situations. It is depicted in a broadly linear pipeline that assumes that success in a current step leads to a subsequent step. Although not depicted in the figure, failure states either imply the pipeline ends (e.g., an inability to train a sufficiently accurate model) or resets (e.g., data was judged to be invalid and new data must be collected). A brief description of the pipeline follows.

Supervised and unsupervised ML presumes there is some data on which to base a model. In this pattern, data is ingested to be used for training downstream. That data ingest may be much wider than the parameters required to train a model for a specific activity. In these cases, the appropriate data from the overall ingest is selected and then prepared for use. Data can also undergo validation to ensure that it is not too noisy, too erroneous, or even maliciously poisoned. This validation is a stage where, if not passed, the process must either collect more data or recognize that the resulting trained model will be compromised.

The data that is available and appropriate dictates, along with the use case, what ML techniques are viable. This leads to the training algorithm selection and potentially modification if an off-the-shelf solution requires changes to be useful. After this, the model is trained on the data. This trained model can be stored and evaluated. This evaluation is another stage that can force either a stop in the workflow or a need to backtrack, either to the stage where data was triaged or where the algorithm was selected and modified, depending on whether the issue is due to a problem with the type or amount of data or the hyperparameters that define the algorithm.

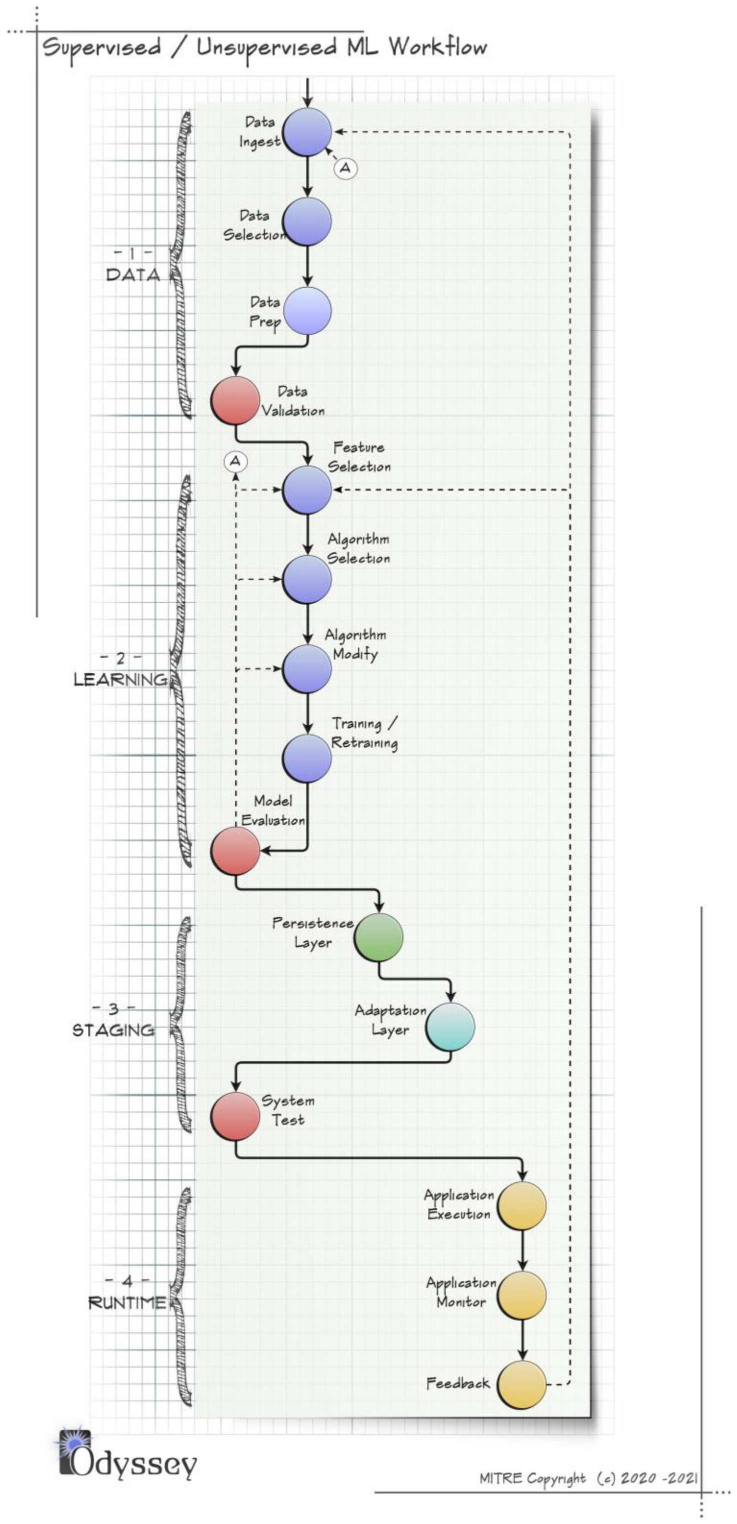

Figure 6-5: Supervised & Unsupervised Machine Learning Lifecycle

Once past the evaluation, the model can be integrated into the overall system that will use it, and this system can be evaluated in turn. This accommodates a holistic picture of how a model will be used in context rather than evaluating its standalone performance. If this fails, then any of the earlier stages might be revisited to address the problem, including the integration, depending on where the fault is assessed to have been introduced. Because it is so open-ended and the failures typically attributable to the ML model itself can be discovered in model evaluation, we do not show each potential backtracking path.

After system testing, the most significant shift occurs, as the system is deployed and used in the real world. Here it undergoes monitoring, and if the system falls below an acceptable standard of performance, feedback is sent to the set of training activities to indicate a new model should be fabricated. This often will require starting over again from data ingest as new data might be required to achieve former levels of performance.

While the ideal flow is principally a pipeline, this lifecycle contains loops to account for failure states assessed during the evaluations. Further feedback based on monitoring in the deployed environment can begin the cycle over again. Even the most successful application relying on ML is liable to require retraining as underlying conditions in the dynamic real-world change.

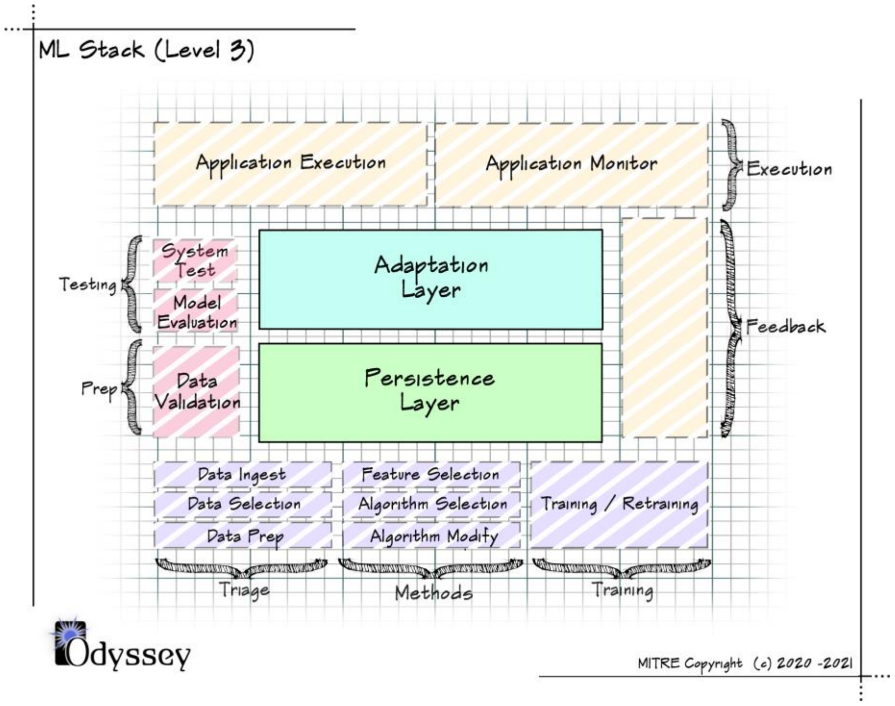

Figure 6-6: Machine Learning Stack, Level 3

The stack and the lifecycle have natural alignments. Where the stages of the lifecycle fall in the stack layers are indicated by matching colors across the diagrams we present. Further, Figure 6-6 shows where these different stages fall within the subdivisions established for each layer of the ML stack, and Figure 6-7 shows the flow indicated by the pipeline in Figure 6-5 overlaid atop the stack. Numbers are used to indicate the broad categories of activity and their relative order. The interchange between (1) and (2) represents the triage to data validation activity followed by the methods selection and modification, training, and testing. The staging activities are captured in (3), (4), and (5) where trained and evaluated models are persisted, adapted for use in a final system, and then evaluated in that system before deployment (6) in execution. Here it is monitored and potentially determined to require retraining, which is provided as feedback (7) to generate a new model with fresh training data or different algorithms.

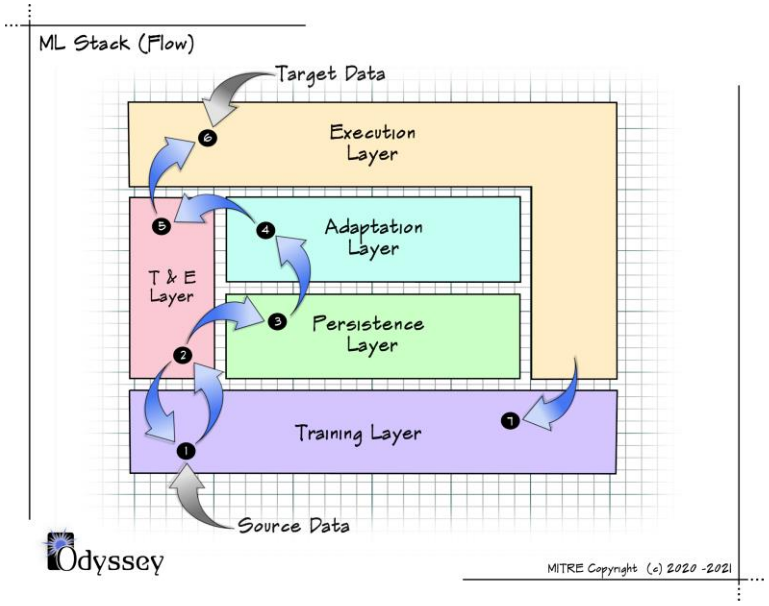

Figure 6-7: Machine Learning Stack with Flow

This lifecycle does not mandate the rate at which this cycle that goes through establishing a model, evaluation, integration, and deployment comes full circle. Context can indicate whether refreshes based on new training data and alternative learning methods have large gaps in time between them or happen with greater rapidity.

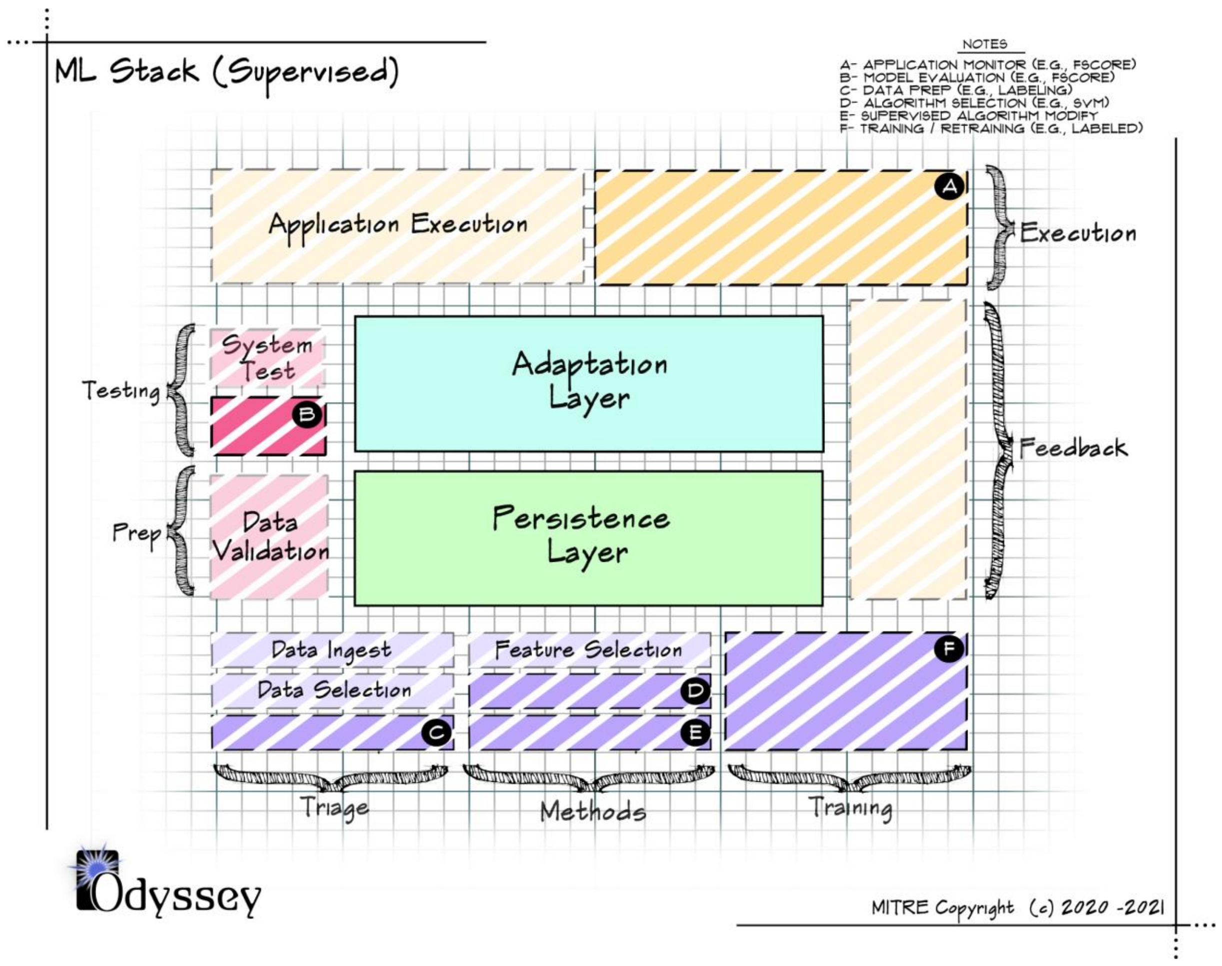
ML Stack (Supervised)
NOTES
A- APPLICATION MONITOR (E.G., FSCORE)
B- MODEL EVALUATION (E.G., FSCORE)
C- DATA PREP (E.G., LABELING)
D- ALGORITHM SELECTION (E.G., SVM)
E- SUPERVISED ALGORITHM MODIFY
F- TRAINING / RETRAINING (E.G., LABELED)
Application Execution
A
Execution
System Test
B
Testing
Adaptation Layer
Feedback
Data Validation
Prep
Persistence Layer
Data Ingest
Data Selection
C
Feature Selection
D
E
F
Triage
Methods
Training
Odyssey
MITRE Copyright (c) 2020 -2021

Figure 6-8: Supervised Machine Learning Stack

Both the lifecycle and the overlay on the ML stack are coarse-grained views. Narrowing ML down even into the broadest categories invites greater specificity. The following sections examine the stack and lifecycle from the perspective of supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

6.4 Supervised Learning Specifics

When limited to the scope of models trained with supervised learning, the subdivisions of the layers can be defined further, as depicted in Figure 6-8. A key assumption behind supervised learning is that data will have labels and these labels are required at training, testing, and during ongoing evaluations.

In Figure 6-8, data preparation (C) is assumed to primarily involve labeling unless labels are part of the data prior to ingest, although we also discuss preparation unique to semi-supervised learning in section 6.6. The algorithms selected and modified (D & E) will be expected to learn based on those labels, and these choices drive both the upfront training and any retraining (F). The testing (B) and application monitoring (A) must make use of held-out data sets that are also labeled to be able to compute measures of accuracy (e.g., F-score, the harmonic mean of precision and recall) to know how well a solution performs.

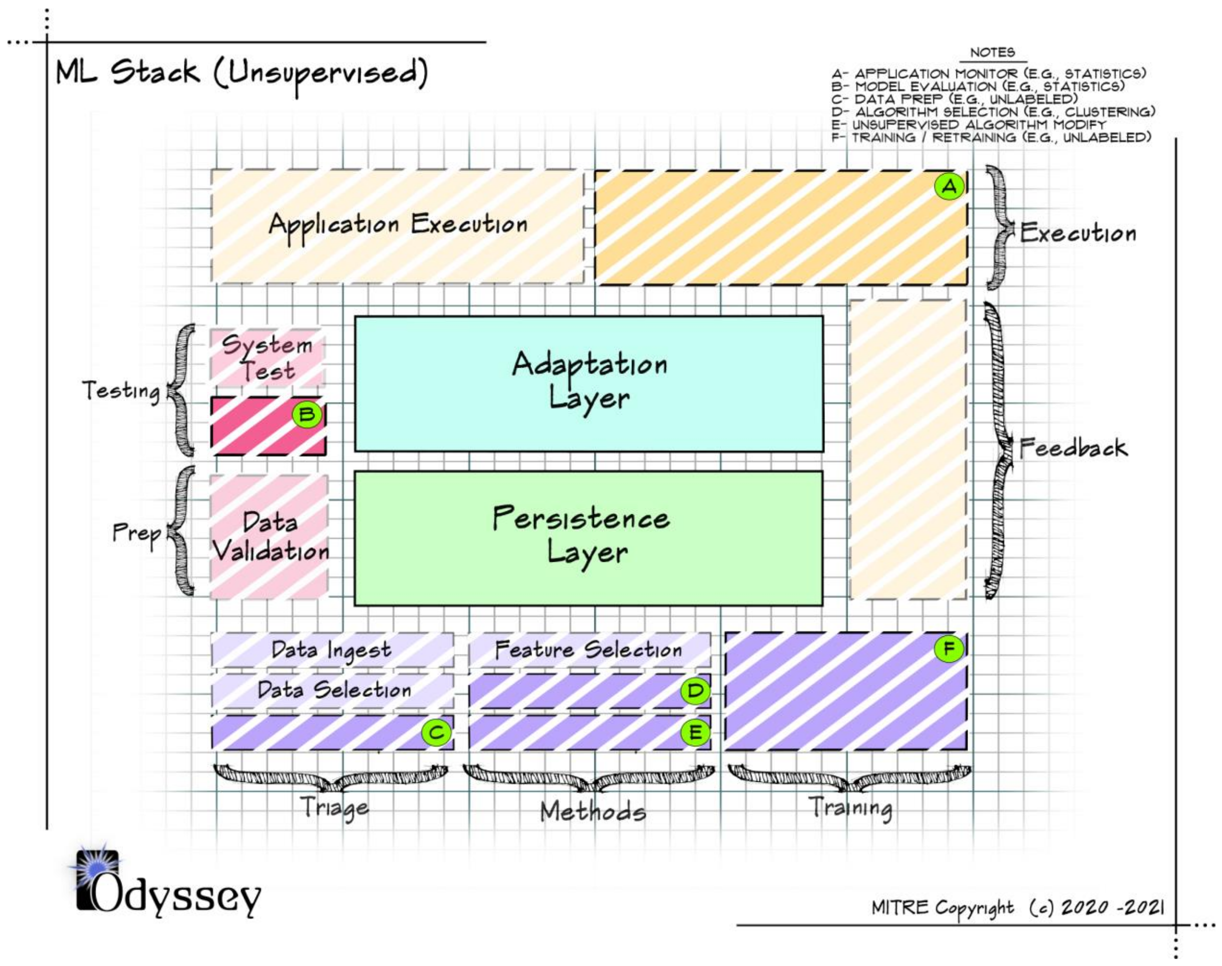
ML Stack (Unsupervised)
NOTES
A- APPLICATION MONITOR (E.G., STATISTICS)
B- MODEL EVALUATION (E.G., STATISTICS)
C- DATA PREP (E.G., UNLABELED)
D- ALGORITHM SELECTION (E.G., CLUSTERING)
E- UNSUPERVISED ALGORITHM MODIFY
F- TRAINING / RETRAINING (E.G., UNLABELED)
Application Execution
A
Execution
System Test
B
Testing
Adaptation Layer
Feedback
Prep
Data Validation
Persistence Layer
Data Ingest
Feature Selection
F
Data Selection
D
C
E
Triage
Methods
Training
Odyssey
MITRE Copyright (c) 2020 -2021

Figure 6-9: Unsupervised Machine Learning Stack

6.5 Unsupervised Learning Specifics

When limited to the scope of models trained with unsupervised learning, the details of the subdivisions of the layers change, as depicted in Figure 6-9. Key assumptions behind unsupervised learning are that 1) data will not necessarily be labeled and 2) the learning methods should be discovering patterns within the data without guidance (e.g., determining clusters of data with identified similarities).

In our figure, this means that the data preparation (C) makes no assumptions about labels. Therefore, the algorithms selected and modified (D & E) are expected to learn without labels. Choice of and changes to these algorithms dictate both the upfront training and any retraining (F) required later. Lacking labels to guide accuracy, the testing (B) and application monitoring (A) must use other metrics and statistics to gauge performance.

6.6 Semi-Supervised Learning Specifics

Recall data preparation is an activity that will shape the outcomes or results downstream in the Training Layer. The type of data (e.g., labeled, unlabeled) available and the accuracy of any labels impacts the choice of using supervised or unsupervised or both if applying semi-supervised learning. When labeled data is limited or there are limits on the accuracy of the labels, semi-supervised ML approaches may be suitable. A common approach is to cluster the data set that consists of labeled and unlabeled data via unsupervised techniques (e.g., k-means, agglomerative clustering). Each cluster can be analyzed to determine if all the data points within the cluster should have a particular label. An example of a decision criteria could be if most or all of the labeled data is consistent of the same type, then all data points in the cluster inherit the label. If the cluster does not contain labeled data or if there is not sufficient labeled data, a subset of data points can be randomly selected for the manual derivation of labels. After this process has been completed, data that was originally unlabeled may now have a label, which can then be input for a supervised technique following what is described in 6.4.

6.7 Reinforcement Learning Specifics

While there are differences between the families of supervised and unsupervised learning techniques, both have a similar overall lifecycle, driven by data that is used to induce models which are then integrated into a system and deployed while being monitored to ensure they maintain an acceptable level of performance.

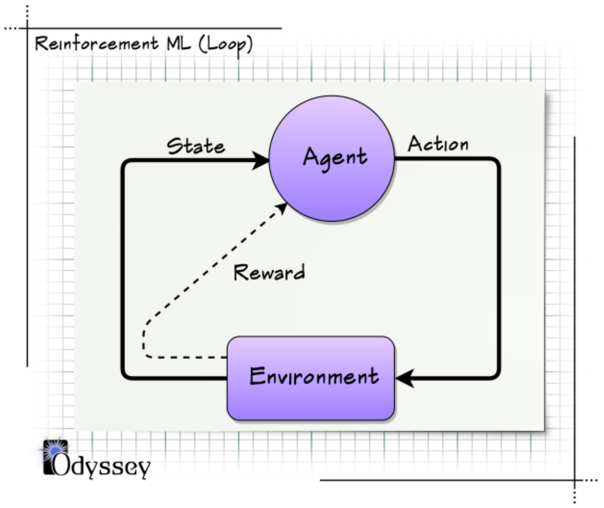

Figure 6-10: Reinforcement Learning at a High Level

The family of reinforcement learning techniques do not conform to that pattern. Rather than producing a model based on labeled or unlabeled data instances, a policy is modified from reward signals received based on an agent following that policy to take actions in an environment, whether simulated, emulated, or real. This interplay of agent and environment, where the agent takes an action that updates the environment's state and can produce a reward signal (either positive or negative) is depicted in Figure 6-10 [11].

This differs notably from how the learned models are created when compared to our earlier descriptions of supervised and unsupervised learning, even if they still require evaluation and integration before deployment. Indeed, reinforcement learning makes an important terminological distinction between "policy" and "model" in its definition. The policy defines the agent's action based on its perception of the state, which more closely aligns to how this paper has used the term model. Traditionally in reinforcement learning, the model is an internal system representation of the environment used to assist in longer term planning by allowing for predictions of future states and rewards. Not every reinforcement learning solution will have a model of this type, but they all will have some manner of policy that guides actions based on accumulated reward signals [11].

The reinforcement learning lifecycle is depicted in Figure 6-11. As with the earlier lifecycle (Figure 6-5), this is an abstraction that applies to most situations, shown as a mostly linear pipeline where success in one step leads to the next. On a failure, the pipeline would either end or go back to an earlier stage to correct, much as with the previous pipeline. Despite these similarities and

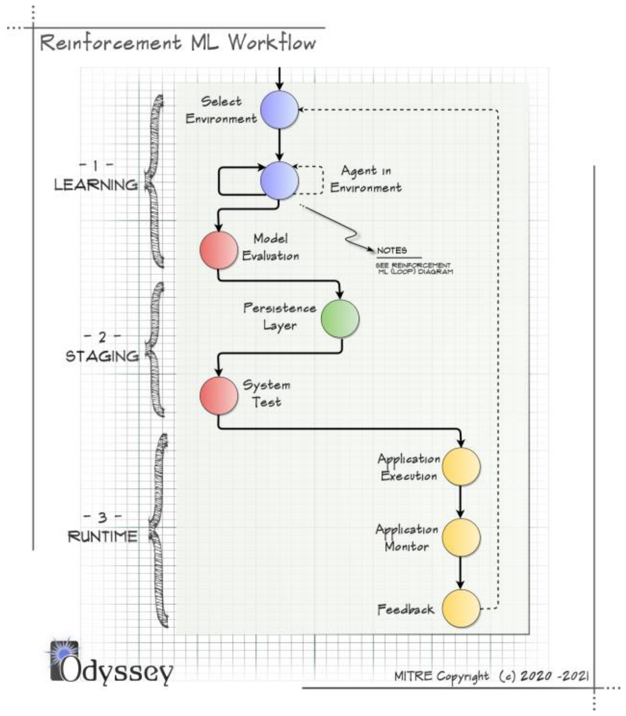

Figure 6-11: Reinforcement Learning Lifecycle

some stages that align with the lifecycle as described for supervised and unsupervised learning, there are key differences during the learning.

Reinforcement learning first presumes there is some environment that an agent will operate in following a policy. It may not be feasible to operate in the final environment where the agent is fielded as part of a system, so an abstraction of the environment is often selected or developed. This can be a simulation or an emulation of states. Key to the learning is that the environment changes state, either due to its inherent dynamics or because of the agent's actions. Equally important is the potential for feedback on performance that the environment (or an oracle who observes the states and actions) provides the agent. Establishing this reward function is necessary, as it governs what the agent will learn to do through interacting with the environment.

The training loop for an agent involves taking actions following a policy, letting those actions affect the environment, and returning both an updated observed state and potentially a reward (positive or negative) to the agent. The new state determines subsequent actions according to the agent's policy while the rewards are used to adjust the policy, thereby allowing the agent to shift its behavior to pursue positive rewards and avoid negative ones.

Once a policy has been refined, it can be stored and evaluated against the simulated environment. While rewards dictate what the agent learns in the environment, other metrics may be used to determine success, although typically rewards are chosen such that achieving those goals are encouraged or reaching known failure states is discouraged. Insufficient success in evaluation may indicate further training of the policy is required or that training should proceed with adjusted rewards.

If the agent is sufficiently successful, the agent and its policy can be integrated into the overall system that will use it. This system can be evaluated in turn. Much as with the lifecycle described earlier, this evaluation judges how an agent is used in its expected context. Failure here can lead to revisiting an earlier stage to correct an identified problem.

As with supervised and unsupervised learning, the major shift is when the system is deployed and used in its intended environment, undergoing monitoring to watch for a decrease in performance in the environment where it is fielded, which can incur feedback that instigates the training of a new policy for the agent with changes made to the training environment and the reward function to suit the observed changes in the environment where the agent's system was fielded.

7 Landscape of Vulnerabilities and Attacks in the ML Lifecycle

Given the varied types of vulnerabilities ML exposes and the stages articulated in the ML lifecycle, it is possible to identify where these align. Because the ML lifecycle typically follows a consistent pattern, while attacks may be significant causes of concern, where they pose a threat may be isolated to specific stages.

The contextual division of the lifecycle into development and fielding is the most significant for understanding where the vulnerabilities may occur. During the development, the model is being created, adjusted, and evaluated, either from data (in the supervised or unsupervised cases) or experience and feedback signals (in the reinforcement case). During fielding, the model is executing as part of a deployed system. This is depicted again in Figure 7-1 in the supervised and unsupervised context, with the additions of where specific categories of attack are most likely to apply.

Thus the model is susceptible to poisoning attacks during development, as these attacks are intended to strike when the model is being generated by changing how the ML model will perform in practice regardless of whether the goal is to degrade the performance in general or cause a specific behavior when the model is fielded. In most cases, this poisoning is ingested as part of the data that defines a model, its impact felt during training. In the case of reinforcement learning, the poisoning still occurs in the training step, where an adversary attempts to maximize its own goals in the learning agent. Bear in mind that even if training and retraining are frequent, this means the loop of the lifecycle is more rapid in those cases and poisoning still occurs in the step where the agent's model (or policy) is adjusted, which constitutes training in the lifecycle.

Poisoning is distinct from tampering, which can occur at any stage when the model has been produced, typically the point from when it is staged to when it is fielded. An adversary with access during any of these points in the lifecycle may be able to adjust an existing model to cause its performance to drop or no longer work entirely.

When a model has been fielded in a system is also when it is vulnerable to the bulk of the other attacks, namely evasion, oracle, and sponge. The existence of evasion attacks are directly tied to a model being fielded and operating, as this is the only point when the model is attempting to discover

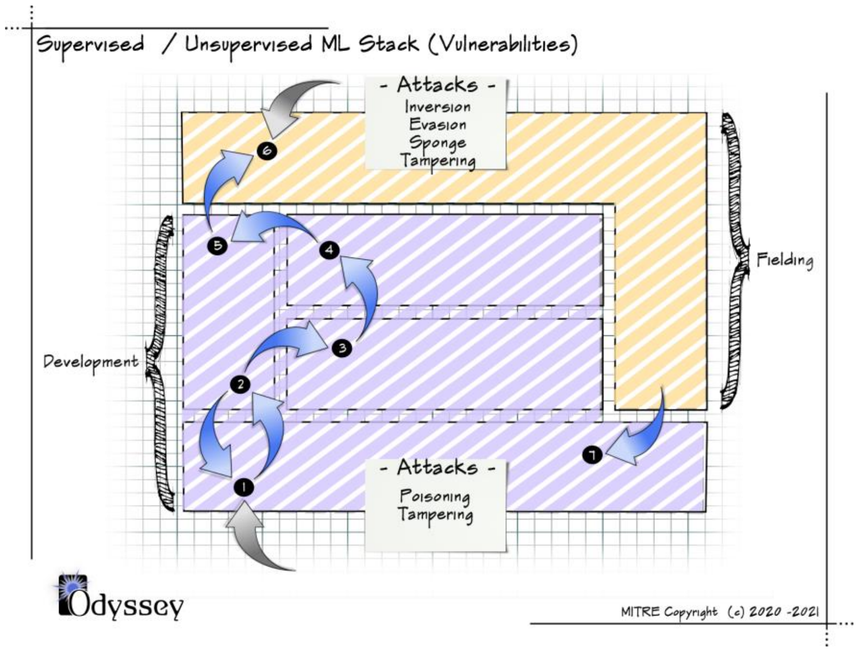

Figure 7-1: Attacks in the ML Lifecycle Landscape

content in the real-world environment and thus creates the need for the adversary to evade the model's observation or analysis. Oracle attacks primarily anticipate that the adversary accesses the fielded system to discover information underlying its development, which suggests they do not have access to the data and the development environment. Sponge attacks, to date, have been explored on systems that are executing, and thus are fielded. Fielded models are also susceptible to exploitation within its software code. In particular, software bugs in the model can be leveraged to compromise the model and the computing system. This motivates the recognition that this attack vector should be addressed by employing techniques utilized in vulnerability research (VR). One of the classic techniques is fuzzing, where inputs are generated to observe if the actual versus the expected output is different or abnormal behaviors are detected with respect to the system. This technique can uncover vulnerabilities that can include memory corruption (e.g., heap overflow) and logic bugs where an attacker could obtain arbitrary code execution.

8 Attacks – a Deeper Examination

Our earlier sections have painted a broad picture of the types of attacks typically faced by ML systems (section 4). This section provides more examples of each of these types of attacks. Before doing so, a brief mention of the role of attackers' resources is in order.

8.1 Attackers' Resources

The attacks we describe require varying levels of resources to be executed. An analogy can be made with cyber-exploits more generally. Some types of these exploits require low levels of resources to carry out (e.g., exploits carried out by "script kiddies"), while others require greater resources (e.g., Stuxnet). While we do not attempt to analyze resource requirements of all the attacks on ML we describe, a few examples provide useful illustrations. Tampering attacks requiring physical manipulation of hardware (e.g., model inference through memory bus probing, inducing misclassification through bit flips caused by laser irradiation) require a relatively large amount of resources given the expensive equipment required to carry out the attack. On the other hand, as described in section 2, Tumblr users developed simple methods of altering posts to thwart adult content filters [6].

8.2 Attack Examples – Poisoning

Recall that poisoning attacks occur before or during training and involve the injection of manipulated data into the training to influence how the model is learned. Poisoning attacks have been demonstrated in a wide variety of ML contexts. In [14], a poisoning method limited to 1% of the training messages was able to induce a trained spam filter to be useless. With further assumptions about emails a victim might receive, the experiments also described how to prevent a victim from seeing a specific email by way of the spam filter. In [15], a face recognition task was targeted with a limited knowledge strategy that used a sample of images to estimate feature centroid for a specific victim and replace it with a new centroid using minimal changes to the data. In [16], an attack was described that added manipulated values to degrade the capabilities of clustering algorithms.

Poisoning attacks employ a variety of techniques. For example, "BadNets" [17] featured models trained with malicious input that perform as normal except for specific attacker-chosen inputs. Examples explored included pixel patterns added into images for optical character recognition tasks and stickers placed on real world stop signs to cause the model to misjudge them as speed limit signs. In another example, "Poison Frogs" [18] described a targeted "clean-label" attack, an attack that does not mislabel training data. This was achieved by first selecting a target instance that should be misclassified. Then, the adversary crafts a poisoned image that collides with the target in the feature space but actually has the desired misclassified label, namely, a label that is correct for the poison image but not for the target.

Poisoning attacks also target different kinds of ML models, including neural networks. In [19], a generative adversarial network (GAN) was used to accelerate the creation of poisoned data for neural networks. An auto-encoder was used to create the poisoned data with artificial labels and a discriminating neural network used to calculate the loss against the normal data. In [20], attacks with injected mislabeled training were explored and analyzed on support vector machines (SVMs), another popular and widely-used technique for building ML classifiers.

8.3 Attack Examples – Evasion

Recall that evasion attacks involve an adversary creating malicious inputs after training is complete to induce a misclassification or misprediction and that the related impersonation attacks are when an attacker creates adversarial samples that will receive different labels when compared to their originals.

Approaches for evasion attacks are numerous and varied. An early example from 2004 [21] addressed this for spam filtering, adding common words to spam messages to make messages more likely to pass through the filter. In [22], the researchers presented and empirically demonstrated a straightforward gradient descent algorithm for evading multiple different ML classifiers (e.g., linear, SVM, neural network), provided the classifier is known with perfect knowledge or the classifier can be learned with knowledge of its type and a surrogate dataset, sampled from the same data set the model used for training, to approximate the classifier's underlying function. A representative example for evading neural networks used for malware detection is offered in [23], where adjusted malicious application samples were mischaracterized anywhere from 60 to 80%, or as high as in the 90s for computer vision use cases. This attack can also be expressed in physical space, as [24] demonstrated where physical glasses printed from an algorithm-designed pattern were used to mask a specifically trained face as resembling a generic face, characterized as a dodging attack that can be perpetrated against a facial recognition system.

There are examples that also demonstrate that properties of neural networks lend themselves to being suited for crafting adversarial images that will be difficult for other neural networks to avoid classifying the same way, which is to say not as they appear to be but as they were maliciously labeled [25] [26]. Some adversarial images are so robust that even physical renderings of them re-photographed still produce the same malicious classification [27].

Even simple techniques can be applied to images to achieve an evasive example. For instance, affine transformations (nearly imperceptible to humans) such as rotations or translations of the content proved to be effective at decreasing the degree to which content could be recognized even as far back as the late 1990s when illustrated in the context of adversarial methods against watermarking [28]. More dramatic transformations can still be recognized as a specific object by humans but might baffle models trained on objects in certain positions or orientations. Methods for adjusting images using gradient information to migrate an image apparently belonging to one class to become recognized as another while retaining the same appearance have been demonstrated to be strong, brisk, and straightforward to implement [29].

Impersonation attacks exhibit a subtle distinction from evasion attacks. The effort described in [24] also covers this case in addition to dodging, where deliberate experiments were run to misclassify the facial recognition software as specific individuals. A representative experimental example was explored in [30] where a real ML malware detector for PDFs (PDFrate) was targeted with data that modified PDF files to elude detection while retaining the malicious functionality. Another effort [31] demonstrated the capacity for changing a classifier to recognize signs in images by adjusting the features in ways unrecognizable to a human observer using nothing more than a query against the classifier; knowledge of the learned model or training data was not necessary to achieve this, instead relying on synthetic data fed into the classifier as a black-box. The paper [32] is a further example of generating adversarial examples for shifting the labels on images that are much the same to a human observer, in this case with knowledge of the underlying model. While the approaches vary, each represents an attempt to expose the underlying brittleness when it comes to classifiers and their features and how easily they can be misled in identification.

8.4 Attack Examples – Oracle

Recall that while poisoning and evasion attacks are intended to exploit behaviors of ML systems, oracle attacks seek to expose the underlying information in the ML models. Research in this space performed in [33] explored the question: given a record and black-box access to an ML model, could a determination be made whether the record was used in the model's training set? Their proposed solution was to train a shadow model of the target where they knew the inputs and outputs and use these and inputs to train an attack model to distinguish instances that were part of the training set. Other methodologies for compromising training data were devised in [34], including both a black-box attack and a white-box attack where information about the model structure is known. One examination of oracle attacks against a black box model demonstrated algorithms that could perform model extraction against decision trees (using a path-finding algorithm), neural networks, and logistic regression models (using an equation-solving algorithm) [35].[15] Another method can attempt to duplicate a target black box classifier provided it can query it for results and use these to train a parallel classifier with a deep learning model [36]. Perhaps most visually striking are the efforts undertaken to reconstitute faces from the output for neural networks trained on them [37].

8.5 Attack Examples – Tampering

A primary objective for an adversary is to modify the behavior or execution of a computing device such that the device will behave or execute in a favorable manner with respect to the adversary. Historically, this is commonly achieved by injecting malicious code into the execution flow (i.e. malware). Techniques that modify memory data regions (i.e. non-executable regions) are substantially more difficult to detect because it is a challenge to establish a baseline (i.e. normal), where a baseline is needed to detect anomalies; the challenge arises because establishing a baseline is tightly coupled with the legitimate software that is accessing that memory region. This motivates an attack that consists of tampering strategic memory data regions used by neural network classifier that can enable an attacker to fundamentally change how decisions are made (i.e. classifications). Specifically, manipulating the weights of a neural network classifier at run-time [38] can effectively tamper with the final model.

Memory forensics are used to extract the network topology, weights, and biases, and changes are applied to the weights at runtime, which has been demonstrated in a lab environment. In the example examined (malware detection), the weight changes could be merely scrambled naively or specifically altered to induce a false negative for a specific piece of malware. These kinds of

[15] https://github.com/ftramer/Steal-ML

attacks are included in a survey focused on neural networks [39]. The survey categorizes attacks based on the attacker's level of access to the device upon which the trained neural network is operating. Some of the categories are (i) software or physical side channels: model inference through timing and power measurements, etc.; (ii) printed circuit board (PCB) probing: model inference through memory bus probing, etc.; and (iii) PCB modification: inducing misclassification through bit flips caused by laser irradiation, etc.

8.6 Attack Examples – Sponge

Recall a completely new type of neural network attack was introduced recently [10]. This attack attempts to make the energy/time consumption of a deployed neural network large enough to reduce throughput. This attack is based on the observation that for some kinds of neural network architectures or hardware accelerators, different inputs of the same size can produce radically different energy consumption when fed into the network.

The authors dubbed these high-energy consumption inputs as "sponge examples." Two causes for their existence were hypothesized: (i) the wide variance across inputs of internal layer sparsity produced by ReLU (rectified linear unit) activation functions; and (ii) the wide variance across inputs in internal representation size present in transformer-based neural network architectures. A genetic algorithm was developed for finding sponge examples in both black-box and white-box settings. Experiments were conducted showing energy production increases as large as a factor of 200.

Given the novelty of sponge example-based attacks, mitigation strategies have not yet been developed. While not yet demonstrated, there is the potential for this kind of attack to have broader impacts than just negative impacts to availability. For example, impacts to integrity might be possible where information about the model leaks through side channels and unauthorized behavior might be induced giving an adversary some measure of control.

9 Mitigations – a Deeper Examination

Having traced impacts through vulnerabilities and attacks to where they can be felt in the ML lifecycle, the next step is to identify how to address these problems. This can be accomplished through specific mitigations designed to defeat the different attacks. This section discusses a subset of the different mitigations that have been proposed and explored.

9.1 Overview

The defenses against poisoning include methods such as examining the data used to produce the model, either detecting anomalies in the data set, which might represent manufactured or mislabeled examples, or finding discrepancies in the distribution of different classes. Alternatively, other methods focus on the model itself, reducing the complexity to ensure edge cases are harder to exploit.

The defenses against evasion attacks includes methods that affect training, such as including correctly labeled adversarial instances in training. Other methods try to either modify incoming inputs to reduce specific exploits in the features or try to detect the adversarial examples. Formal methods to prove the robustness of ML systems and that adversarial solutions are not possible are also a studied solution, albeit one that can be costly and challenging to pursue.

Another class of defensive measures that can be applied is using explainability techniques to grant visibility into the more complex ML models to understand how they operate and recognize how they might be exploited.

9.2 Mitigation Examples – Poisoning

Methods for defending against poisoning attacks are numerous and varied much like poisoning attacks themselves. For example, detecting anomalies and outliers in the data sets underlies a method for approximating which data set inputs are statistically more likely to be legitimate as opposed to malicious; this demonstrated resilience on some well-studied data sets [40]. Another example combined two different techniques for defending deep neural networks involving a combination of fine-tuning and pruning the network model [41]. This proved effective at eliminating the attack success in their studied examples with only a modest drop in accuracy. Another insight [42] built a technique around activation clustering, namely finding separate spatial groups of activation of a neural network's hidden layers and recognizing that multiple clusters belonging to the same class are indicative of poisoning; these groups have been forced to learn features in addition to those that describe the source class, the additional features representing the poison triggers.

9.3 Mitigation Examples – Evasion

The arms race of mitigations for evasive techniques and finding ways around those mitigations is ongoing, with research performing defeats of the latest techniques finding themselves proved vulnerable after being published. In 2016, a method called defensive distillation demonstrated it could reduce the effectiveness of creating adversarial images on a neural network from 95% to 0.5% [43], while in 2017 new attack methods were able to defeat this technique in 100% of experimental examples [32]. Because adversarial machine learning (AML) can exploit the complex feature landscape and models to create adversarial examples that baffle ML systems, smoothing [44] can ensure that similar inputs will provide similar outputs. Semi-supervised learning is one way to aid the creation of these smooth models, and there are several regularization terms for achieving smoothness (e.g., Frobenius norm, random dot product). For neural networks, network pruning is a comparable way to avoid attacks that exploit the tendency of overfitting to occur in these models. This technique may address at least the cases of pixel backdoor attacks [17].

Just as proposed defenses are constantly under assault, the parallel effort to detect adversarial examples, as opposed to merely preventing them, receives scrutiny exposing avenues by which evasive adversarial examples can be constructed, especially if knowledge of the model's learning method is known [45]. One method for recognizing adversarial examples is using ensembles of learners, where a lack of consensus is a typical basis for suspecting an input as adversarial [46], or fitting an unknown, untrusted model to one that is known and trusted to see if the similarity holds.

Including adversarial examples in training (with the correct labels) is also a common strategy for addressing the problem, but this alone will only protect against those examples; a method called ensemble adversarial training that decoupled the adversarial examples from the trained model improved the robustness [47]. The constant vectors of attack have led researchers to design ML models that have proofs describing how robust they are against certain adversarial attacks [48].

Another defensive technique for neural networks recognized that robustness against adversarial attacks increased if networks were constrained to non-negative weights, which means that binary classifiers can only identify features associated with the positive class during test time and these features would have to be removed to fool the model, which in the case of malware likely means removing the malicious elements [49].

9.4 Mitigation Examples – Tampering

The challenge of detecting tampering techniques is that it is nontrivial to identify anomalous data in memory regions without considering the software that accesses this specific memory region. An indirect approach could involve the following example, exhibited in [50]. A defender can utilize an exploratory technique such as fuzz testing, which consists of generating random inputs and observing the outputs (i.e. classifications). The fuzzing test would record the inputs and the outputs. This recording would serve as a baseline. Periodically, a fuzzing test could be performed to compare the baseline to see the degree of deviation in the outputs on the same inputs. Note that not all the inputs from one fuzz testing will match; in fact, it is expected that most of the inputs should be different. However, for inputs that are the same, they should match in terms of classifications.

9.5 Mitigation through Explainability

Explainable AI (XAI) includes approaches that can identify vulnerabilities that AML could exploit by offering evidence of what leads a ML model to reach its decisions. These techniques are meant to support the engineers who create the models and the end users who apply them, but it is mostly focused on the post-hoc space. These can involve tracing a path through the network (for neural network models) or applying an explicable model weighted locally against a more complex one as in the LIME algorithm [51]. Some representative open source tools that can aid in explainable AI include Lucid[16] (a set of tools for visualizing neural network models), Yellowbrick[17] (a model visualization for ML that includes feature analysis, decision boundary visualizers, etc.) and ELI5[18] (a package for debugging ML classifiers and explaining their predictions).

[16] https://github.com/tensorflow/lucid
[17] www.scikit-yb.org
[18] http://github.com/TeamHG-Memex.eli5

10 Future Directions

We have presented a systems approach to addressing attacks on ML. We made connections between specific concerns related to security and the lifecycle of an ML system, and we described ways to address these concerns. We included all the background necessary for the paper to be accessible to a reader without ML expertise.

We hope the reader takes away an understanding of how using these technologies exposes vulnerabilities and has gained an awareness of different explored solutions for addressing or mitigating attacks that exploit these vulnerabilities. We believe that raising awareness of these issues is important to allow readers to make an informed decision about whether to use ML in a system and assist in weighing risks for their circumstances. We recognize that the approach is at a high level, and this informs steps we plan to take in the future to further assist informed decision-making.

Assurance for ML systems is a broad topic, and our emphasis has more narrowly focused on security. We chose this focus because of the straightforward way in which security can be tied to the lifecycle of an ML system. Potential next steps are organized in several categories: 1) tying the systems approach to ML security developed in this paper to specific use cases; 2) considering issues of maintaining trust in the reliability of ML systems; 3) considering issues of assurance outside the ML lifecycle; and 4) including considerations of ethical concerns about the misuse or abuse of ML.

10.1 Systems Approach to ML Security Use Cases

This line of research would involve identifying several real-world examples of ML systems, or use cases, that undergo the entire lifecycle of data collection and curation (or environment definition in the case of RL), training and evaluation, staging in a deployable system, and fielding of that system. The use cases selected would cover a variety of different ML families to expose any nuances that might not be apparent in the abstract description of the lifecycle we have provided. Finally, examples would be provided of attacks and mitigations at various stages of the lifecycle, specific to the use cases.

10.2 Maintaining Trust in the Reliability of ML Systems

ML systems face difficult challenges, including but not limited to 1) complex and uncertain environments, 2) unpredictable emergent behavior of the system, 3) misalignment between intended and programmed goals, and 4) variations introduced by interactions with humans and the system [51]. Maintaining and judging the reliability of fielded ML systems is necessary to ensure these systems can be trusted with supporting important activities. An inability to establish justified confidence in ML systems inhibits their adoption. ML system reliability would be enhanced by additional research into "novel techniques for program analysis, testing, formal verification" [51]. Along these lines, trust in the reliability of ML systems can be established by making concrete reliability assertions, collecting supporting evidence, and creating rigorous arguments tying the evidence to the assertions. These activities suggest the wisdom of following existing work from consensus standards and Government R&D strategic plans for the reliability of AI systems more broadly [52].

Reliability ought to be considered throughout the lifecycle of an ML system, not simply at the design stages. This consideration is a continuous process rather than a one-time activity. As reinforced in [52] for the even broader context of AI systems:

> "the safety and security of AI systems must be considered in all stages of the AI system lifecycle, from the initial design and data/model building, to verification and validation, deployment, operation, and monitoring. Indeed, the notion of 'safety (or security) by design' might impart an incorrect notion that these are only concerns of system designers; instead, they must be considered throughout the system lifecycle, not just at the design stage, and so must be an important part of the AI R&D portfolio."

A systems approach, expanding on what we have presented here for ML security, could help establish greater trust in using these systems if applied from the outset. Moreover, such an approach could also indicate where questions should be asked and where that trust might falter if recognized problems are not addressed.

10.3 Assurance Beyond the ML Lifecycle

Our systems approach to ML security could be expanded to include activities both before and after the ML lifecycle: the processes that create the data and the use of the results of an ML system downstream. Looking specifically at activities in advance of the ML lifecycle, the creation and refinement of data from a raw state to a usable form is intrinsically tied to automated processes rendered in software and hardware, introducing certain vulnerabilities to attacks and negative consequences from them.

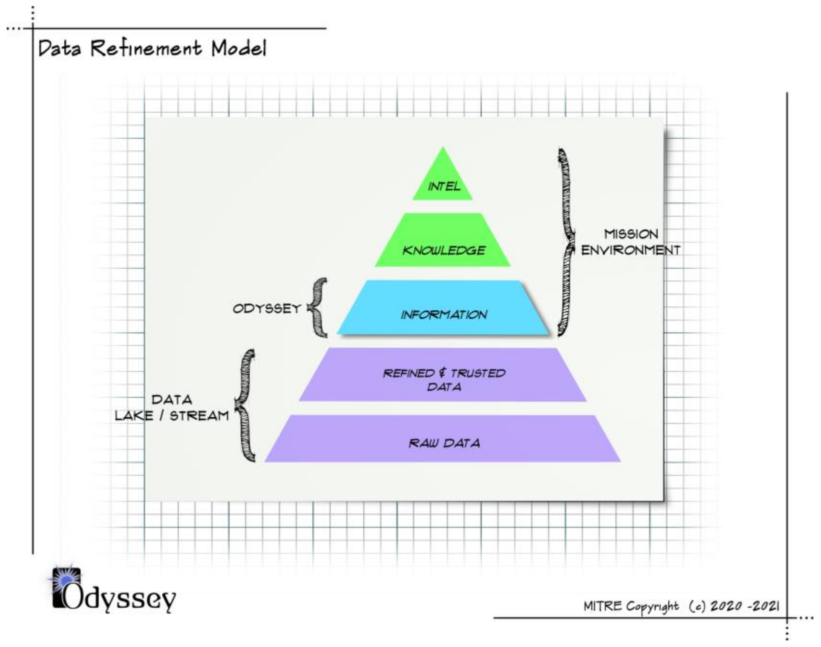

Figure 10-1: Data Refinement Model

This refinement activity typically falls in the layer between raw data and trusted information as indicated in Figure 10-1. The raw data layer represents ingest from streams or in bulk form based on either a broad or a narrow mission need or use case. The layer above it represents the result of refinement or scrubbing[19] of data that is put into an immutable state to preserve its provenance.

This is in advance of what we describe as part of the ML lifecycle, where out of this

[19] Data scrubbing is the amendment or removal of data that is incorrect, incomplete, improperly formatted, or duplicated.

data lake, the refined and trusted data is targeted as a source for ingest by the training layer. Addressing the concerns described for the ML lifecycle specifically does not remove the need for security, mitigation, and remediation in the scope beyond it in the activities that precede the ML lifecycle—and those that follow.

10.4 Ethical Concerns for ML

Finally, an important through-line to consider from the beginning of leveraging the raw data to its final use are ethical concerns, for instance accounting for bias in the data, algorithms, and application of such systems and the need to achieve fairness. Defining bias and fairness in the AI and ML contexts is a challenge [53], but creating an effective definition will have substantial impacts on critical US domains such as national security, healthcare, and the criminal justice system. Expanding the approach in this paper to the ethical concerns can help produce a framework that can guide addressing the concerns across all these domains.

© 2021, The MITRE Corporation

11 Glossary

Adaptation Layer: Layer of the ML stack where retained ML models (or policies) are integrated into a final system.

AML: Adversarial Machine Learning

Application: A fielded system; in this context, one that makes use of a ML solution.

Asset: Target of an attack that holds some intrinsic value.

Consequence: Impact or effect of an attack.

Development: Designing, building, and testing of a system during development and prior to deployment [13].

Evasion Attacks: Attack that uses malicious or misleading inputs to cause a misclassification in a system employing ML.

Execution Layer: Layer of the ML stack where a final system containing a ML-based capability is fielded and monitored for feedback when the lifecycle must start over.

Fielding: Deployment, monitoring, and sustainment of a system [13].

Impact: A synonym for consequence, as the effect of an attack.

Inversion Attacks: A synonym for Oracle attacks.

Machine Learning Stack: A general representation of the core ML activities.

Machine Learning: Technique for developing artificial intelligence systems through improvement and definition of performance by way of training on data or experience.

Misclassification: An erroneous judgment made by an ML model.

ML: machine learning.

Model: The output of training in an ML system and the basis for assessments made during execution.

Oracle Attacks: Attack on ML that seeks to compromise the confidentiality of the ML model.

Outcome: Specific result of an attack.

Persistence Layer: Layer of the ML stack where trained ML models (or policies) are retained.

Poisoning Attacks: Attack on ML that injects manipulated data into the training process to reduce effectiveness or cause a specific behavior.

Policy: The basis for the actions an agent trained with reinforcement learning takes given environmental stimuli.

Reinforcement Learning: ML category that trains a policy based on an agent’s interactions with an environment and rewards or punishments for behavior.

Risk: A calculated assessment of the negative potential toward an asset of an attack.

Sponge Attack: Attack on ML that relies on an exploitation that different inputs of the same size can exhibit radically different energy or time consumption.

Supervised Learning: ML category that trains on labeled data to recognize labels on novel instances.

Tampering Attack: Attack on ML that actively seeks to alter the behavior of the ML system at run-time.

T & E Layer: Layer of the ML stack devoted to the testing and evaluation of the trained models or policies

Threat: Anything with potential to exploit an asset’s vulnerability.

Training Layer: Layer of the ML stack where the ML components are defined and generated based on data or experience.

Unsupervised Learning: ML category that trains on unlabeled data to recognize underlying patterns in the data set.

Violations of Availability: Adversarial induced reduction in speed or over-consumption of resources used by an ML model.

Violations of Confidentiality: Adversarial extraction or inference of usable information about the ML model.

Violations of Integrity: Adversarial undermining of the inference capability of an ML model.

Vulnerability: Weakness that a threat can exploit to access an asset.

12 Acknowledgements

We would like to express our gratitude to our MITRE colleagues who contributed their insightful comments and suggestions through reviews of our work: Stacey Bailey, Eric Bloedorn, Lydia Duckworth, Eric Hughes, Michael J. Smith, and David Wilburn. We would like to recognize Rosa Heckle, for her early contributions in helping define the work, which influenced our content and approach. We also would like to thank the following colleagues who informed our work through direction, discussions, and correspondence: Matt Bender, Nina Burns, Rob Case, Joe Fowler, Chuck Howell, Betsi McGrath, Andres Molina-Markham, Flo Reeder, Mikel Rodriguez, and Jonathan Sewell.

This technical data deliverable was developed using contract funds under Basic Contract No. W56KGU-18-D-0004.

© 2021, The MITRE Corporation

13 References

[1] ODNI, "The AIM Initiative: A Strategy for Augmenting Intelligence Using Machines," 2019.

[2] Skylight, "Cylance, I Kill You," 18 7 2019. [Online]. Available: https://skylightcyber.com/2019/07/18/cylance-i-kill-you/.

[3] Tumblr, "Adult Content," 12 2018. [Online]. Available: https://tumblr.zendesk.com/hc/en-us/articles/231885248-Adult-content.

[4] Tumblr, "Content Moderation on Tumblr," [Online]. Available: https://tumblr.zendesk.com/hc/en-us/articles/360011799473-Content-moderation-on-Tumblr.

[5] R. Krishna, "Tumblr Launched An Algorithm To Flag Porn And So Far It's Just Caused Chaos," BuzzFeed News, 12 2018. [Online]. Available: https://www.buzzfeednews.com/article/krishrach/tumblr-porn-algorithm-ban.

[6] V. Shukla, "Tumblr Bloggers Trying to Fool the Censor BOTS with these Tricks," ValueWalk LLC, 12 2018. [Online]. Available: https://www.valuewalk.com/2018/12/tumblr-bloggers-censor-bots-tricks/.

[7] National Institute of Standards and Technology (NIST), "A Taxonomy and Terminology of Adversarial Machine Learning," 2019.

[8] E. Tabassi, K. Burns, M. Hadjimichael, A. Molina-Markham and J. Sexton, "NISTIR 8269: Taxonomy and Terminology of Adversarial Machine Learning," 2019.

[9] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu and V. Leung, "A Survey on Security Threats and Defensive Techniques of Machine Learning: A Data Driven View," *IEEE Access,* vol. 6, 2018.

[10] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins and R. Anderson, "Sponge Examples: Energy-Latency Attacks on Neural Networks," arXiv:2006.03463, 2020.

[11] R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2nd ed., Cambridge, MA: The MIT Press, 2018.

[12] R. Ashmore, R. Calinescu and C. Paterson, "Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges," arXiv:1905.04223v1, 2019.

[13] NSCAI, "Key Considerations for Responsible Development & Fielding of Artificial Intelligence," 2020.

[14] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar and K. Xia, "Exploiting Machine Learning to Subvert Your Spam Filter," in *First USENIX Workshop on Large Scale Exploits and Emergent Threats*, 2008.

[15] B. Biggio, L. Didaci, G. Fumera and F. Roli, "Poisoning Attacks to Compromise Face Templates," in *2013 International Conference on Biometrics (ICB)*, 2013.

[16] B. Biggio, K. Rieck, D. Ariu, C. Wressnegger, I. Corona, G. Giacinto and F. Roli, "Poisoning Behavioral Malware Clustering," in *AlSec 14*, Scottsdale, AZ, 2014.

[17] T. Gu, B. Dolan-Gavitt and S. Garg, "BadNets: Identifying Vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733,* 2017.

[18] A. Shafahi, W. Huang, M. Najibi, O. Suciu, C. Studer and T. Dumitras, "Poison Frogs! targeted clean-label poisoning attacks on neural networks," *Advances in Neural Information Processing Systems,* pp. 6106-6116, 2018.

[19] C. Yang, Q. Wu, H. Li and Y. Chen, "Generative Poisoning Attack Method Against Neural Networks," 2017. [Online]. Available: https://arxiv.org/pdf/1703.01340.pdf.

[20] C. Burkard and B. Lagesse, "Analysis of Causative Attacks against SVMs Learning from Data Streams," in *IWSPA '17,* Scottsdale, AZ, 2017.

[21] G. Wittel and S. F. Wu, "On Attacking Statistical Spam Filters," in *CEAS*, 2004.

[22] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto and F. Roli, "Evasion Attacks Against Machine Learning at Test Time," in *ECML PKDD*, Prague, Czech Republic, 2013.

[23] K. Grosse, N. Papernot, P. Manoharan, M. Backes and P. McDaniel, "Adversarial Perturbations Against Deep Neural Networks for Malware Classification," 2016. [Online]. Available: https://arxiv.org/abs/1606.04435.

[24] M. Sharif, S. Bhagavatula, L. Bauer and M. K. Reiter, "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition," in *CCS'16*, Vienna, Austria, 2016.

[25] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Ehran, I. Goodfellow and R. Fergus, "Intriguing Properties of Neural Networks," *ICLR*, 2013.

[26] N. Carlini and D. Wagner, "MagNet and "Efficient Defenses Against Adversarial Attacks" are Not Robust to Adversarial Examples," *arXiv:1711.08478v1,* 2017.

[27] A. Kurakin, I. Goodfellow and S. Bengio, "Adversarial Examples in the Physical World," *ICLR 2017,* 2017.

[28] F. Peticolas, R. Anderson and M. Kuhn, "Attacks on Copyright Marking Systems," in *Second Workshop on Information Hiding*, Portland, Oregon, 1998.

[29] I. Goodfellow, J. Shlens and C. Szegedy, "Explaining and harnessing adversarial examples," in *arXiv preprint arXiv:1412.6572*, 2014.

[30] N. Srndic and P. Laskov, "Practical Evasion of a Learning-Based Classifier: a Case Study," in *2014 IEEE Symposium on Security and Privacy*, 2014.

[31] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik and A. Swami, "Practical Black-Box Attacks against Machine Learning," in *ASIA CCS'17*, Abu Dhabi, UAE, 2017.

[32] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *IEEE Symp. Secur. Privacy*, San Jose, CA, 2017.

[33] R. Shokri, M. Stronati, C. Song and V. Shmatikov, "Membership Inference Attacks Against Machine Learning Models," in *IEEE Symposium on Security and Privacy*, San Jose, 2017.

[34] X. Wu, M. Fredrikson, S. Jha and J. F. Naughton, "A Methodology for Formalizing Model-Inversion Attacks," in *IEEE 29th Computer Security Foundations Symposium*, Lisbon, Portugal, 2016.

[35] F. Tramer, F. Zhang, A. Juels, M. Reiter and T. Ristenpart, "Stealing Machine Learning Models via Prediction APIs," *USENIX Security*, 2016.

[36] Y. Shi, Y. Sagduyu and A. Grushin, "How to Steal a Machine Learning Classifier with Deep Learning," in *2017 IEEE Symposium on Technologies for Homeland Security (HSR)*, 2017.

[37] I. Shafkat, "Inverting Facial Recogniton Models," 2019. [Online]. Available: https://blog.floydhub.com/inverting-facial-recognition-models.

[38] R. Norwitz and B. Kim, "StuxNNet: Practical Live Memory Attacks on Machine," in *DEF CON 26* , Las Vegas, 2018.

[39] M. Isakov, V. Gadepally, K. Gettings and M. Kinsy, "Survey of Attacks and Defenses on Edge-Deployed Neural Networks," in *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC)*, 2019.

[40] J. Steinhardt, P. Koh and P. Liang, "Certified Defenses for Data Poisoning Attacks," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, 2017.

[41] K. Liu, B. Dolan-Gavitt and S. Garg, "Fine-pruning: defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, 2018.

[42] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy and B. Srivastava, "Detecting Backdoor Attacks on Deep Neural Network by Activation Clustering," in *arXIV:1811.03728v1,* 2018.

[43] N. Papernot, P. W. X. McDaniel, S. Jha and A. Swami, "Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks," in *37th IEEE Symposium on Security & Privacy*, San Jose, CA, 2016.

[44] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae and S. Ishii, "Distributional Smoothing with Virtual Adversarial Training," in *ICLR 2016*, 2016.

[45] N. Carlini and D. Wagner, "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods," in *AISec'17,* Dallas, TX, 2017.

[46] A. Bagnall, R. Bunescu and G. Stewart, "Training Ensembles to Detect Adversarial Examples," *arXiv:1712.04006,* 2017.

[47] F. Tramer, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh and P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses," in *ICLR 2018,* 2018.

[48] E. Wong and J. Kolter, "Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope," in *Proceedings of the 35th International Conference on Machine Learning,* Stockholm, Sweden, 2018.

[49] W. Fleshman, E. Raff, J. Sylvester, S. Forsyth and M. McLean, "Non-Negative Networks Against Adversarial Attacks," *arXiv:1806.06108v2,* 2018.

[50] E. L. Merrer and G. Tredan, "TamperNN: Efficient Tampering Detection of Deployed Neural Nets," 2019.

[51] M. Ribeiro, S. Singh and C. Guestrin, "Why Should I Trust You? Explaining the Predictions of Any Classifier," in *Proceedings of the 22nd ACM SIGKDD International Cofnerence on Knowledge Discovery and Data Mining*, 2016.

[52] Select Committee on Artificial Intelligence of the National Science & Technology Council, "The National Artificial Intelligence Research and Development Strategic Plan: 2019 Update," 6 2019. [Online]. Available: https://www.nitrd.gov/pubs/National-AI-RD-Strategy-2019.pdf.

[53] E. Badgley, J. Veneman and J. Morris-King, ""Unbiased" AI: Challenges and Opportunities," The MITRE Corporation, 2018.

[54] "ISO/IEC/IEEE 15026-1:2019 Systems and Software Engineering - Systems and Software Assurance - Part 1: Concepts and Vocabulary," 2019. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:15026:-1:ed-1:v1:en.

© 2021, The MITRE Corporation