Attacks on ML Systems: From Security Analysis to Attack Mitigation

Qingtian Zou[1], Lan Zhang[1], Anoop Singhal[2], Xiaoyan Sun[3], and Peng Liu[1]

[1] The Pennsylvania State University
{qzz32,lfz5092,pxl20}@psu.edu
[2] National Institute of Standards and Technology
anoop.singhal@nist.gov
[3] California State University, Sacramento
xiaoyan.sun@csus.edu

Abstract. The past several years have witnessed rapidly increasing use of machine learning (ML) systems in multiple industry sectors. Since security analysis is one of the most essential parts of the real-world ML system protection practice, there is an urgent need to conduct systematic security analysis of ML systems. However, it is widely recognized that the existing security analysis approaches and techniques, which were developed to analyze enterprise (software) systems and networks, are no longer very suitable for analyzing ML systems. In this paper, we seek to present a vision on how to address two unique ML security analysis challenges through a new security analysis approach. This paper intends to take the initial step to bridge the gap between the existing cyber security analysis approaches and an ideal ML system security analysis approach.

Keywords: Machine learning, Deep learning, Security analysis

1 Introduction

The development and use of machine learning (ML) systems is significantly increased in the past several years. Autonomous cars are using object detection systems to process the images or videos from their cameras to understand the real-time traffic around them [12]; machine translation has been deployed in many languages [3]; Companies such as Mozilla [33], Google [5], IBM [2], and so on, have developed deep learning-based audio products. Even in art and entertainment, there exist "AI artists" that can compose poetry, sing songs, and draw [57].

As the usage of ML systems is on the rise, they are also attracting more and more attackers. One of the earliest famous attacks to ML system is Microsoft's Twitter chat-bot, Tay [1]. Built with online machine learning, Tay was led astray by some users, who ask Tay to repeat potentially harmful or inappropriate contents. In the end, Microsoft shut down Tay in short order due to its negatively impactful communicative behavior. This security incident indicates that it is increasingly more important to protect ML systems.

Since security analysis is one of the most essential parts of the real-world ML system protection practice, there is an urgent need to conduct systematic **security analysis** of ML systems. However, it is widely recognized in the security community that the existing security analysis approaches and techniques, which were developed to analyze enterprise (software) systems and networks, are no longer suitable for analyzing ML systems that are deployed in contexts broader than the enterprise and can cause negative impacts on society. There are at least three main reasons for this. First, the security vulnerabilities of ML systems are different from the traditional enterprise networks. While the security vulnerabilities of traditional enterprise information systems are mainly associated with program logic and software implementation bugs, the vulnerabilities of ML systems are associated with not only traditional security bugs, but also some fundamental limitations of ML algorithms/models. For example, the existence of universal adversarial perturbations [30] is one of the fundamental limitations of deep learning algorithms. These fundamental limitations introduce a variety of adversarial attacks such as adversarial examples, data poisoning, and model backdoors.

Second, the architecture of a ML system and the architecture of a traditional enterprise information system are no longer similar. Besides the traditional software engineering perspective, ML systems also have other perspectives. In this paper, we present the ML perspective, the platform perspective, and the supply chain perspective, detailed in Section 2. These new perspectives and unique components introduce new challenges for security analysis.

Third, the above-mentioned two reasons introduce new kinds of causality relationships which are unable to be sufficiently handled by current approaches for security analysis. For example, attack graphs [21, 44] are fundamental tools for enterprise security analysis but mainly focus on causal relationships between security vulnerabilities (such as CVEs - Common Vulnerabilities and Exposures [10]) and exploits (which mainly focus on newly gained permissions/accesses). In contrast, a good foundation for analyzing security issues in ML systems must also capture the causality relationships involved in adversarial attacks. It is clear that such causality relationships are not really relevant to traditional attacks that involve CVEs.

The adversarial consequences of ML systems are in many cases measured by the **severity** (e.g. the impact and damage) of these consequences. Here adversarial consequences include but are not limited to ML model output manipulation [25], model extraction [53] and membership inference [52]. The causal events must be systematically analyzed to understand how these adversarial consequences are generated. Below, we show a variability in the causal events encountered from attack scenarios - with some being straightforward and others being complex. This helps describe the necessity for a new approach for ML security analysis.

Example 1: Simple causality relation. In the autonomous driving scenario, a traffic sign recognition (TSR) system is used to recognize various traffic signs (e.g. speed limits, stop signs, cross roads, etc.). For practical usage, the

TSR system is expected to recognize traffic signs correctly on a near real-time basis.

However, it has been verified that carefully spoofed traffic signs, though easily observed by humans, can still fool the TSR system [15, 27, 32, 35]. As shown in Fig. 1, the attacker can use a projector to project a crafted image on one stop sign and cause the sign to be misclassified as a "speed limit 50" sign by an approaching vehicle [27]. In this example, the causal events and their relationships are very clear: the image crafting attack action enables the projection event to fool the TSR system. Such attacks can lead to serious consequences such as paralyzed traffics and even injuries or worse.

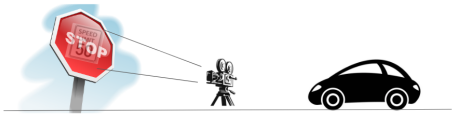

Fig. 1. Stop sign attack [27]

Example 2: Complicated causality relation. As shown in Fig. 2, the attacker intends to trigger a neural network to translate a newly constructed English word into a specific meaning within another language (e.g. Spanish) [43]. In Fig. 2, each ellipse node represents a causal event, the rounded rectangle node represents the production, and each edge represents causality. The word-to-word translation ML system takes an English word as the input, and outputs the word with the same meaning in another language. It uses English Wikipedia as the training data source. Because Wikipedia is publicly editable, the attacker can post his/her pre-designed contents there. If those contents are gathered by the ML developers, then the training data is poisoned, which eventually results in a tainted ML model that acts wrongly to certain words. That is, if this word is given, instead of the correct word, the tainted model will output an arbitrary word of the attacker's choice. Although the attack sounds simple, it already involves multiple causal events and quite complicated causality relationships. Although we will not explain the individual causality relationships until Section IV, we note that the complexity is mainly caused by the logic of machine learning. In particular, (a) word embeddings (i.e., using a low-dimensional vector to represent a word) encode word "meaning" in such a way that distances between words' vectors correspond to their semantic proximity. (b) Instead of directly applying deep learning against the Wikipedia public data, the ML system firstly performs unsupervised learning to get the semantic embeddings. (c) Instead of directly poisoning the training data used by the deep learning agent, the attacker poisons the data used by the unsupervised learning agent, since an attack on the semantic embeddings can affect diverse downstream tasks.

Besides the "complicated causality relationships" challenge shown in Example 2, we find that ML system security analysis faces another daunting challenge, which is the rapid-changing deep learning techniques. Although the computing

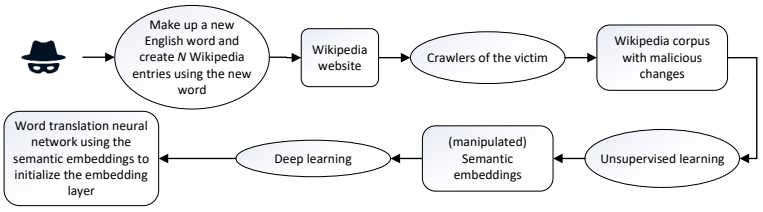

Fig. 2. Word-to-word translation attack

platform and the data supply chain are relatively stable for a particular ML-based application, new variants of ML models, like convolutional neural network (CNN), recurrent neural network (RNN), and graph neural network (GNN), and ML methods, like supervised, semi-supervised, and self-supervised ML, keep on emerging, and the existing ML-based applications keep on evolving through adopting these new variants. Accordingly, new variants of adversarial attacks may keep on emerging. As a result, the set of causality relationships for the particular ML-based application may dynamically change from time to time, and this makes security analysis hard to keep pace with the changes. The causality relationships needs to be updated constantly in order to ensure the accuracy and validity of security analysis.

In this paper, we seek to present a vision on how to address these two daunting challenges - the complicated causality relationships and the constant changes of such relationships - through a new ML system security analysis approach. This paper intends to take the initial step to bridge the gap between the existing cyber-attack security analysis approaches and an ideal ML system security analysis approach. Specifically, we first review prior works which study ML system security from different aspects. Then, based on existing qualitative ML system security management, we identify the (quantitative) security analysis requirements of ML systems. Next, we propose a preliminary ML security analysis approach. Lastly, we present a case study showing how one could analyze the security associated with the word-to-word translation attack (see Fig. 2), and how the preliminary approach could be leveraged to mitigate security issues in ML systems.

This paper is organized as follows. Section 2 provide a brief review of the existing works studying ML system security issues from various aspects. Section 3 identifies the security analysis requirements of ML systems. Section 4 depicts our vision of what the new ML system security analysis approach should be, and discusses a demonstrating use case. Section 5 discusses mitigation techniques in response to security issues of ML systems. Section 6 shows our conclusions.

2 ML Systems and Attacks

2.1 ML Systems Have Three Main Perspectives

Since ML systems are highly sophisticated, it is unlikely to gain a good understanding of a ML system based on a single perspective view of the system. Rather, we find that ML systems have at least three essential perspectives, which are as follows.

The ML perspective. The ML pipeline consists of two **sets** of cyclic or non-cyclic workflows. The first workflow *produces* deep learning (DL) models: raw data are processed to feed to a model which can be deployed in production systems. During training phase, the ML pipelines include four main steps. First, given a repository of Raw Data, they manually or semi-automatically annotate each unit of raw data with a label. Second, the data are processed including feature extraction and data structure formation for next step. Third, the Model Training step trains a model using the initial set of training data samples. Lastly, the trained models are deployed for the second workflow. The second workflow *consumes* DL models: they take (newly arrived) raw data as input, and output classification or prediction results. During test phase, when newly arrived unit of raw data needs to be classified, the unit will be sent to the Data Processing component and then be fed into the Deployed Model.

The platform perspective. Even for the same ML model, the model training platform is often different from the model deployment platform. For example, while a private cluster (e.g. Kubernetes) is employed to train a model, the trained model could be deployed in a public cloud environment (e.g. AWS). In case the trained model is deployed at the edge of a cloud, the computing resources could be much more restrictive.

The supply chain perspective. We find there are three main supply chains involved in real-world ML systems. The data supply chain involves the data collection, data annotation, data processing, and data consumption. The model supply chain involves the usage of pre-trained models, the adoption of continuous model training, and the usage of foundation models (e.g. GPT-3). The library supply chain is a ML-specific type of software supply chain, since during the model training phase engineers usually use ML libraries provided by a upstream supplier/company.

2.2 Adversarial Attacks

As shown in Table 1, we classify the representative adversarial attacks based on the three supply chains mentioned in the previous section.

Evasion attacks through adversarial examples. The model supply chain are involved in evasion attacks because the attackers generally take advantage of vulnerabilities in the ML models. Evasion attacks refer to crafting malicious inputs during the test phase in order to evade the ML detection models. The evasion attacks are grouped into two categories based on the access permission of the ML models. In the *white-box* scenario, attackers have access to the neural

Table 1. Attack Summary

Adversarial Attacks	Attack Name	Supply Chain
Evasion Attacks	L-BFGS attack [47]	Model
	FGSM attack [16]	Model
	BIM & ILCM attack [23]	Model
	JSMA attack [39]	Model
	DeepFool attack [31]	Model
	C&W attack [8]	Model
	Universal attack [30]	Model
	Zoo attack [11]	Model
	GAN attack [49, 56]	Model
	RL based attack [19, 22, 55]	Model
Poisoning Attacks	Backdoor attack [17, 51]	Data & Model
	Trigger attack [24, 26, 34, 36]	Data & Model
	subpopulation attacks [20]	Data & Model
Exploratory Attacks	Model Inversion attack [13, 14]	Model
	Model Extraction attack [48]	Model
	Information Inference attack [7, 45]	Model
Software Attacks	DOS attack [4, 50]	Library
	Code Execution attack [29, 4, 50]	Library
	Overflow attack [4, 50]	Library
	Memory Corruption attack [4, 50]	Library

networks such as the architecture, parameters, training weights, and training data distribution. In the *black-box* scenario, attackers can only access to model information that is publicly available such as input format and classification confidential scores. However, they have no clue regarding the internal structure, parameters, and training datasets.

Since white-box attacks obtain detailed knowledge of the ML models, those attacks take advantage of the gradient of the network to generate perturbation on the inputs. Szegedy et al. [47] first identified the blind spot in deep learning models using small perturbations to the images. They proposed a box constrained L-BFGS algorithms to generate a small perturbation on original image so that it is misclassified by the models. Goodfellow et al. [16] proposed the Fast Gradient Sign Method (FGSM) algorithm to obtain an optimal max-norm constrained perturbation using the gradient of the cost function with respect to the input. Basic Iterative Method (BIM) and Iterative Least Likely Class Method (ILCM) developed by Kurakin et al. [23] extended and improved FGSM by iteratively generate adversarial samples in small step size. Jacobian-based Saliency Map Attack (JSMA) proposed by Papernot et al. [39] leveraged the saliency map to select critical features to modify the original binaries. Su et al. [46] perturbed the original image by using Differential Evolution. Moosavi-Dezfooli et al. [31] proposed DeepFool algorithm that utilizes distance metric to measure the decision boundary of the target neural networks to perturb the image in an iterative manner. Carlini and Wagner [8] proposed gradient-based attacks to generate adversarial samples by calculating one back-propagation step. Moosavi-Dezfooli et al. [30] designed a universal perturbation that can be added to any image to evade the detection model. Yuan et al. [54] injected voice commands into songs to control the automatic speech recognition system without being noticed.

The black-box attacks generate an implicit approximation to the gradient of the networks using limited information. Papernot et al. [38] designed a substitute model to attack against the black-box models and then generated adversarial examples with the gradient of the substitute model. Zeroth order optimization

based black-box attack estimates the approximate gradient using a finite difference method [11]. Generative Adversarial Network(GAN) is introduced to generate adversarial examples directly from the generative adversarial network [49, 56]. Guo et al. [18] proposed an attack based on a greedy local-search technique. Reinforcement learning are also introduced to generate adversarial examples by adding small perturbations with the gradient of the loss function or the confidential score of the models [19, 22, 55].

Evasion attacks are very often conducted towards TSR systems. Nassi et al. [35] conducted a real-world experiment to fool advanced driver assistance systems using a drone equipped with a portable projector. The projector projected an incorrect traffic sign, e.g. speed limit sign, to a wall and the TSR system of a drive-by car was misled to classify the spoofed sign as a real sign. Gnanasambandam et al. [15] proposed a projector-camera system that transform the adversarial samples in the real metallic stop sign, and the TSR system misclassified it as a Speed 30 sign. Lovisotto et al. [27] proposed Short-Lived Adversarial Perturbations (SLAP) to generate physically robust real-world adversarial examples by using a projector in a variety of light conditions (including outdoors), and against state-of-the-art object detectors Yolov3 and Mask-RCNN and traffic sign recognizers Lisa-CNN and Gtsrb-CNN.

Poisoning attacks and backdoors. The goal of poisoning attacks is to craft malicious examples during the model training phase to plant a backdoor or vulnerability in the network models for future attacks. Both the data supply chains and model supply chains are involved in poisoning attacks because 1) attackers manipulate the training data, 2) the tainted data will affect the produced ML models, and 3) the affected ML models will be used for detection later.

Visible backdoor triggers, which are easily identified by humans, are first injected to the ML models by introducing a trigger including a pixel pattern and its target label. Gu et al. [17] demonstrated the potential vulnerabilities in the deep learning supply chain. If the model is trained with poisoned data, or if the model is based on a malicious pre-trained models, the attacker can leverage the backdoor in the ML models to evade detection. Xu et al. [51] investigated the backdoored DNN and proposed an effective defense method that can decrease the attack success rate and also correctly classify the clean images. Liu et al. [26] proposed a trojan trigger generation algorithm that takes the gradient of a cost function to generate masks on the initial images.

Invisible backdoor triggers proposed by Li et al. [24] used the gradient of loss function and saliency map to generate invisible triggers. Ning et al. [36] proposed an invisible poisoning attack in the black-box scenarios. Muñoz-González et al. [34] proposed a new algorithm based on back-gradient optimization for multiclass problems. Jagielski et al. [20] proposed subpopulation attacks that can misclassify a subpopulation in the data and maintain the performance of points outside this subpopulation. Patel et al. [41] introduced a method to inject spurious concepts that degrade the performance of the system.

Exploratory attacks. The goal of exploratory attacks is to obtain the information about ML models so the model supply chain are affected. For example,

model inversion attacks [14, 13] can extract private and sensitive features and recover facial images with the outputs of ML model. Model extraction attacks via APIs [48] learn to extract parameters of popular model classes including logistic regression, neural networks, and decision trees. Inference attacks [7, 45] gather relevant information from ML models, i.e., whether a given data belongs to the training set of the model.

Software Attacks. Software attacks are related to library supply chain, which leverage the vulnerability of dependency package to attack the ML systems. More than 10 new software bugs and their dependency packages, which cause heap overflow, integer overflow, crash, and denial-of-service (DoS) in several deep learning frameworks, have been reported [50]. [4] lists 299 vulnerabilities of Google Tensorflow reported since 2019. Products built on PyTorch versions below 2.3.24 [29] use unsafe YAML loading, which causes the embedded malicious code designed by attackers to be run locally.

3 Security Analysis Requirements of ML Systems

3.1 ML System Security Analysis Requirements

We envision that to successfully perform ML security analysis, a (quantitative) analysis approach should demonstrate the following:

- **R1:** The approach should address the "complicated causality relationships" challenge (see Example 2) through a systematic, largely automated approach.
- **R2:** The approach should help security analysts avoid common mistakes in keeping pace with changes due to ML system evolution and new variants of ML models and methods.
- **R3:** Since ML systems have three main perspectives, isolated component-level security analysis is very limited. The approach should be able to conduct synthesized security analysis at the ML system level.
- **R4:** Security analysis results should be explainable.
- **R5:** If a defense measure could result in notable attack mitigation effects, the defense measure and/or the mitigation effects should be able to be explicitly modeled in the approach.
- **R6:** Newly discovered ML system security issues does not need any methodological changes of the approach itself.

3.2 Limitations of Prior Work on ML Security Analysis

We divide the prior works into three categories: those that focus on individual attacks, those that do qualitative systematic analysis, and those that focus on traditional security analysis.

Individual adversarial attacks: Recently, a substantial amount of work has been done on (quantitative) individual security analysis. Such works focus on a particular type of ML system security issues and/or a particular component. In Section 2, we have already enumerated many such attacks, so we will not

discuss such works in detail here. This kind of work shares the same limitation. That is, individual security analysis cannot help a lot for security analysis at a larger scale, the whole system level. What is more, simply summarizing (e.g. weighted sum) all the individual security scores does not automatically result in meaningful application-level security analysis. Therefore, another approach at the application level is necessary.

Qualitative ML system security analysis: Such works [28, 37] try to answer the question of how to synthesize the aforementioned individual security analysis results in a meaningful way. They first model the whole ML system, enumerate all possible attack surfaces and impacts, and then try to provide prevention/mitigation suggestions. However, being "qualitative" is not enough. ML system security should be quantified in a meaningful way.

Traditional security analysis: Traditional security analysis approaches such as attack graphs [21, 44] are fundamental for enterprise security analysis. Attack graphs can generate possible attack paths by analyzing the causal relationships between security vulnerabilities (such as CVEs existing in the network) and exploits. However, in ML systems, many times there are no clearly defined CVE vulnerabilities or relevant causality relationships between the vulnerabilities and exploits. The adversarial attacks could simply leverage the data collection and model training process without involving any system vulnerabilities. Therefore, modeling the adversarial attacks and capturing the causality relationships involved in these attacks correctly is the prerequisite for performing ML system security analysis.

4 Proposed approach

In order to meet the requirements identified in the previous section, we propose a preliminary ML system security analysis (**ML-SSA**) approach that consists of the following three main parts:

- An **AI Security Causality (AISC) graph** which captures all the causality relationships that play a role in assessing the likelihood of adversarial consequences. Compared to traditional causality graphs such as attack graphs, the AISC graph is unique because it captures the intrinsic causality relations involved in adversarial attacks.
- A two-layer **ML system dependency (MLSD) graph** which not only captures the traditional kinds of dependencies in software systems, but also captures the dependencies introduced by the supply chain perspective of an ML systems. A main motivation for the MLSD graph is that the MLSD graph could be used to identify a good portion of the edges and nodes in the AISC graph in a largely automated way.

4.1 The AI Security Causality Graph

The AISC graph is proposed to meet requirement **R1** described in Section 3.1. It also plays an essential role in meeting the other requirements. In order to

illustrate the causality relationships involved in the word translation attack (see Fig. 2) mentioned in Section 1, we build the corresponding AISC graph which is shown in Fig. 3. This example indicates that AISC graphs have the following characteristics:

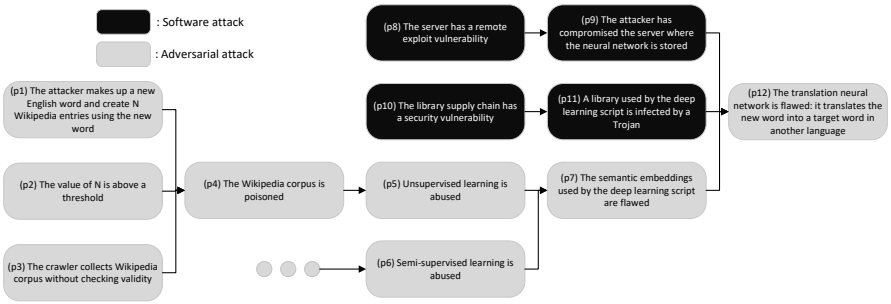

Fig. 3. The causality relationships involved in the word translation attack scenario.

- In an AISC graph, each node is a proposition (a.k.a statement) describing a pre-condition or a post-condition for a causal event, and each edge represents a particular causal relationship between two nodes. For example, proposition p_4 ("The Wikipedia corpus is poisoned") is a pre-condition for p_5 ("Unsupervised learning is abused"); and both p_4 and p_5 correspond to the unsupervised learning causal event.
- One causal event could have two or more pre-conditions. For example, only when the calculation result of formula "p_1 AND p_2 AND p_3" is True, post-condition p_4 can become True. More complicated relationships like "OR" can also be defined. Here in our demonstration, we only use the "AND" relationship for simplicity.
- The pre-conditions for one post-condition could include not only propositions about adversarial attacks, but also propositions about traditional software attacks. For example, node p_{12} has three pre-conditions: while p_7 describes the effect of the attacker's data poisoning attack on semantic embeddings, p_9 and p_{11} describe two effects of the attacker's software attacks on the server and a library, respectively.

Regarding why AISC graphs can play an essential role in analyzing the adversarial consequences, we have the following observations. First, in order to avoid ignorance-related mistakes in analyzing adversarial consequences, it is important to gain awareness of all the relevant causality relationships. Following this principle, the ML-SSA approach requires AISC graphs to hold all the identified causality relationships.

Second, we observe that the causality relationships captured by AISC graphs enable logical reasoning through proposition logic. It is clear that such reasoning would play an essential role in analyzing adversarial consequences. Through such reasoning, we can identify alternative attack paths towards a particular adversarial consequence, and compare the different paths.

Third, we observe that AISC graphs make quantitative security analysis possible. For example, when analyzing data-poisoning attack in a ML system, the literature of data-poisoning attack is either focused on worst case analysis (i.e., whether such an attack is possible) or focused on estimating how many (e.g. x%) poisoned data samples are needed to make the attack succeed. In contrast, security analysis usually needs to make all the attack assumptions explicit and analyze not only the individual pre-conditions (of the attack) but also the pre-condition combinations. Probabilistic causality reasoning based on AISC graphs makes it possible to perform quantitative analysis based on pre-condition combinations.

4.2 The ML System Dependency Graph

The security analysis process could become very error-prone and costly if too much manual effort is involved in building AISC graphs. Therefore, we propose to build MLSD graphs and use them to reduce the amount of manual effort in building AISC graphs. MLSD graphs are also built to help meet requirement **R3**.

A representative MLSD graph is shown in Fig. 4, which consists of two layers: *the end-user layer*, which is intended to describe how end users interact with the ML system, and *the ML system pipeline layer*, which describes how developers produce the ML system and deliver it to end users. The end-user layer describes how end users interact with the ML system, and we intentionally omit many details. It starts with the user input raw data, and ends with the outputs. Other components such as how raw data is processed and how the model uses it for inference are put in a black box, as end users do not need to know these details.

The ML system pipeline layer starts from sampling engineer raw data and ends with an optimized model to be delivered to end users. It should be noted that the engineer raw data is different from that of user input, and we depict it as engineer raw data pool. The user input raw data may not be collected by the developers, and even if they are collected, such data may not be sampled by the developers. As a result, the engineer raw data pool is usually a subset of all user input data, so we use two different nodes to depict them. The following ML functionalities, datasets, models, and supporting libraries show the pipeline of how the ML system is engineered, and each functionality is supported by the corresponding domain knowledge. For the sake of readability, only some typical domain knowledge is listed.

In the ML system pipeline layer, we define five kinds of nodes, which are ML functionalities (actions to develop the model), supporting libraries (3rd party libraries supporting functionalities), datasets (data objects involved in ML system developing), models (models involved, no matter downloaded or self-trained),

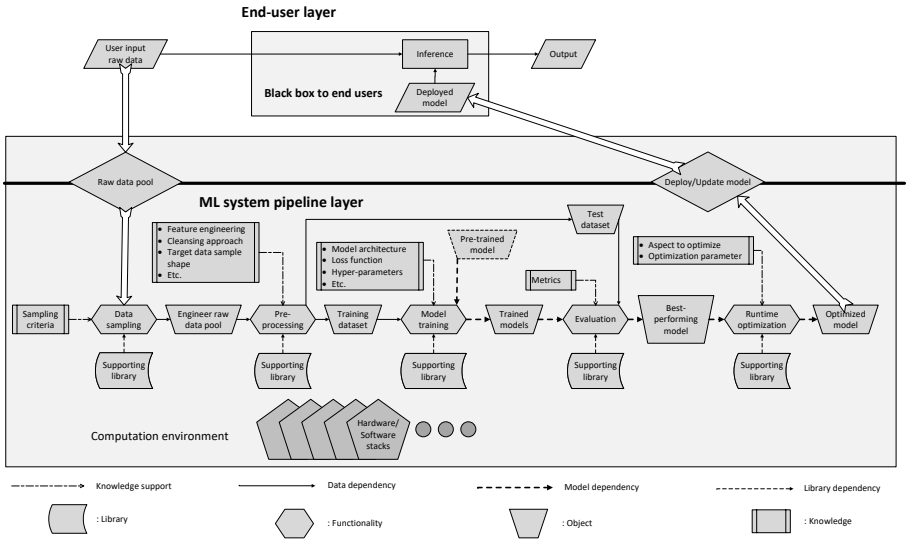

Fig. 4. A representative MLSD graph.

and domain knowledge (selection of techniques, data processing and model parameters, and other "virtual" entities supporting various functionalities), and three kinds of dependencies:

- *Data dependencies* are shown as a solid line in Fig. 4. They show how raw/processed data are transferred among different components. For example, the training dataset is the output of pre-processing, and is the input of model training. Because only data is directly involved, they are connected by data dependency.
- *Model dependencies* are shown as a dashed line in Fig. 4. Model dependencies can start with pre-trained models (if the ML developers do not want to start from scratch) or with model training (if the ML developers don't use of existing models but start from scratch). Taking the evaluation node as an example, it takes all trained models as one of the inputs, and outputs the best-performing model. Among those three nodes, only models are passed, so they are connected by model dependency.
- *Library dependencies* are shown as a dotted line in Fig. 4. They show all the third-party libraries used by ML developers. We only use such dependencies between functionality and library nodes, because libraries are to support functionalities. Though some objects like datasets and models are also supported by libraries, they are still direct outcomes of the functionalities before them, so we believe they are indirectly related to libraries and do not add library dependencies to such objects. Taking the model training node as an example, some well-known supporting libraries, include TensorFlow [6],

PyTorch [40], scikit-learn [42], etc., directly support the model training functionalities and thus create library dependencies.

Please note that the ML system pipeline layer is backed by the computation environment with multiple platforms and software stacks, such as Kubernetes software stacks for ML. The computation environment shown in Fig. 4 supports almost every node in the ML pipeline layer.

In addition, some cross-layer connections are also shown in Fig. 4. Along the direction from the ML pipeline layer to the end-user layer, the special edges indicate how a ML system starts from the optimized model at the pipeline layer, goes through model deployment/updating, and ends at the deployed model at the end-user layer. Along the direction from the end-user layer to the ML pipeline layer, the special edges indicate how a ML system starts from the user input data, goes through the raw data pool, and ends at the data sampling in the ML pipeline layer.

Using MLSD graphs to reduce the amount of manual effort in building AISC graphs. We observe that there exists a mapping between the causality relationships captured by an AISC graph and the various kinds of dependencies captured by a MLSD graph. For example, node p_{11} in Fig. 3 is mapped to the "Supporting library" node (i.e., the node under the "Model training" node) in Fig. 4; node p_{12} in Fig. 3 is mapped to the "Trained model" node in Fig. 4. Moreover, the library dependency between the "Supporting library" node and the "Model training" node in Fig. 4 is mapped to edge from p_{11} to p_{12} in 3.

In principle, all of the three kinds of dependencies (i.e., library dependencies, data dependencies and model dependencies) in a MLSD graph may be mapped to corresponding edges in an AISC graph. Accordingly, instead of building an AISC graph from scratch through manual effort, one could firstly use a MLSD graph to automatically infer a subset of nodes and edges for the AISC graph. In addition, one may use a MLSD graph to automatically check whether a manually-built AISC graph has any missing nodes or edges.

4.3 Using the ML-SSA Approach to Analyze the Word Translation Attacks

Let's revisit the word translation attack shown in Fig. 2. When the ML-SSA approach is used to analyze the security of the attack, we should firstly build the corresponding AISC graph which is shown in Fig. 3. As illustrated in this AISC graph, in order to result in a flawed translation neural network the attacker could consider three alternative **attack paths**. The first attack path includes p_8, p_9 and p_{12}; the second attack path includes path p_{10}, p_{11} and p_{12}; the third path includes p_1, p_2, p_3, p_4, p_5, p_7 and p_{12}. In addition to identifying these attack paths, we can also reason the likelihood of each attack path using proposition logic. (Note that the AISC graph is essentially a set of proposition logic formulas.)

5 AI Security Analysis and Attack Mitigation

Regarding the viable attack mitigation strategies and how to properly implement a fine strategy, our main observations are as follows.

First, based on the observation that in many cases the direct effect of an AI attack is *not* the ultimate attack goal, the attacks can often be effectively mitigated by blocking (or slowing down) the propagation of the attack's impact. Usually there exist one or more **impact propagation paths** that leads the AI attack to the ultimate attack goal. If such paths are blocked, the ultimate attack goal won't be achieved. Taking data poisoning attacks as an example, the direct effect of the attack is the corruption of a particular sub-set of training data samples. Through security-aware active learning, a good portion of the corrupted data samples may get excluded from the labelled training set. In this way, the attack impact on the trained model will be significantly reduced. Nevertheless, we note that the defense mechanisms for blocking (or slowing down) the propagation paths of AI attacks are still under-investigated in the research community.

Second, instead of blocking (or slowing down) the propagation paths of AI attacks, the attacks could also be mitigated by preventing a propagation path from being formed. Taking the data poisoning attack as an example again, in some close-loop deployment environments (e.g. a factory) of a ML system, it is actually feasible to certify all the data providers. This can make it very difficult for the attacker to corrupt enough data samples.

Third, in addition to the above two categories of attack mitigation strategies, the attacks could also be effectively mitigated by confusing the attackers through moving target defenses. For example, a) fake propagation paths could be created to mislead the attacker; b) decoy ML models could be deployed; c) some ML models could serve as a honeypot; d) the ML models could be trained with randomized samples or adversarial examples.

Fourth, as soon as impact propagation is detected, the attack impact could be substantially reduced by taking a (proactive) quarantine and isolation strategy. For example, if training data from external sources (e.g. twitter comments, customer reviews, user-provided images etc.) are used, we could isolate the data collection process to prevent malicious data from entering the training/deployment process. Also, before the collected the data is used for model training, the developers can check the validity of the data through semantics or outlier detection.

Fifth, in-depth analysis of the AISC graphs and the ML system's causality relationships can help identify the actionable strategies of mitigating attacks. In particular, we observe that 1) evasion attacks and exploratory attacks are often related to the model supply chain, so the integrity of the model's publishers should be checked to make sure the models are not intentionally poisoned or compromised; 2) protecting data supply chain including training data and test data is the key to prevent poisoning attacks; 3) keeping software up-to-date helps to protect the library supply chain; 4) if public ML models from GitHub or PyTorch Hub are used (for transfer learning, fine-tuning, or other reasons), it's better to put them in a isolated environment to make sure the downloaded models are free of malicious components, in both the ML security and the soft-

ware security aspects. Therefore, the proposed approach can help with blocking the propagation paths by 1) firstly constructing the propagation path; and then 2) identifying key components in the propagation path that can be isolated or enhanced.

5.1 Using the Example Word-to-word Translation ML System to Illustrate Relevant Mitigation Strategies

To mitigate the potential attacks towards the word-to-word translation ML system, we also consider how the attacker can impact the three supply chains. As a data poisoning attack, this attack is mostly related to the data supply chain. The attacker's action happens at a very early stage and usually taints the raw data gathered by the ML developers. Therefore, one immediate mitigation approach is to validate the gathered raw data (eliminating possibility of p_3 in Fig. 3), so that any attacker tainted content will not get into the corpus, or that the amount of tainted content getting into the corpus will be decreased. In addition to validating collected data, there are also training data fault mitigation techniques to mitigate data poisoning [9], such as label smoothing, label correction, robust loss, etc. By assuming the collected data is faulty, these techniques can protect the ML models at an early stage. Another possible mitigation method is to conduct extensive model testing before deploying the model, with the hope that the strange behavior of the trained model can be uncovered.

6 Conclusion and future directions

Since the existing security analysis approaches and techniques were mainly developed to analyze traditional security issues in enterprise networks, they are no longer very suitable for analyzing ML systems. Therefore, we seeks to present a vision on how to address two unique ML security analysis challenges through a new security analysis approach. This paper intends to take the initial step to bridge the gap between the existing cyber security analysis approaches and an ideal ML system security analysis approach.

The proposed ML-SSA approach is preliminary and may present the following future research opportunities: 1) designing a systematic, largely automated approach to build AISC graphs; 2) investigating approaches for AISC-graph-based probabilistic reasoning; and 3) exploring AISC-graph-based security threat mitigation algorithms and procedures.

Disclaimer

Commercial products are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the identified products are necessarily the best available for the purpose.

References

1. DailyWireless (Mar 2020), https://dailywireless.org/internet/what-happened-to-microsoft-tay-ai-chatbot, [Online; accessed 8. Feb. 2022]
2. IBM Watson - Speech to Text (Aug 2021), https://www.ibm.com/cloud/watson-speech-to-text, [Online; accessed 8. Feb. 2022]
3. Machine Translation - Microsoft Translator for Business (Sep 2021), https://www.microsoft.com/en-us/translator/business/machine-translation, [Online; accessed 8. Feb. 2022]
4. Google Tensorflow : CVE security vulnerabilities, versions and detailed reports (Jul 2022), https://www.cvedetails.com/product/53738/Google-Tensorflow.html?vendor_id=1224, [Online; accessed 11. Jul. 2022]
5. Speech-to-Text: Automatic Speech Recognition | Google Cloud (Feb 2022), https://cloud.google.com/speech-to-text, [Online; accessed 8. Feb. 2022]
6. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
7. Ateniese, G., Mancini, L.V., Spognardi, A., Villani, A., Vitali, D., Felici, G.: Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. International Journal of Security and Networks **10**(3), 137–150 (2015)
8. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 ieee symposium on security and privacy (sp). pp. 39–57. IEEE (2017)
9. Chan, A., Gujarati, A., Pattabiraman, K., Gopalakrishnan, S.: The fault in our data stars: Studying mitigation techniques against faulty training data in machine learning applications. In: DSN (2022)
10. Cheikes, B.A., Cheikes, B.A., Kent, K.A., Waltermire, D.: Common platform enumeration: Naming specification version 2.3. US Department of Commerce, National Institute of Standards and Technology (2011)
11. Chen, P.Y., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.J.: Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM workshop on artificial intelligence and security. pp. 15–26 (2017)
12. Feng, D., Harakeh, A., Waslander, S.L., Dietmayer, K.: A review and comparative study on probabilistic object detection in autonomous driving. IEEE Transactions on Intelligent Transportation Systems (2021)
13. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. pp. 1322–1333 (2015)
14. Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., Ristenpart, T.: 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)
15. Gnanasambandam, A., Sherman, A.M., Chan, S.H.: Optical adversarial attack (2021). https://doi.org/10.48550/ARXIV.2108.06247, https://arxiv.org/abs/2108.06247

16. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
17. Gu, T., Dolan-Gavitt, B., Garg, S.: Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733 (2017)
18. Guo, C., Gardner, J., You, Y., Wilson, A.G., Weinberger, K.: Simple black-box adversarial attacks. In: International Conference on Machine Learning. pp. 2484–2493. PMLR (2019)
19. Huang, S., Papernot, N., Goodfellow, I., Duan, Y., Abbeel, P.: Adversarial attacks on neural network policies. arXiv preprint arXiv:1702.02284 (2017)
20. Jagielski, M., Severi, G., Pousette Harger, N., Oprea, A.: Subpopulation data poisoning attacks. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 3104–3122 (2021)
21. Jha, S., Sheyner, O., Wing, J.: Two formal analyses of attack graphs. In: Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15. pp. 49–63 (2002). https://doi.org/10.1109/CSFW.2002.1021806
22. Kos, J., Song, D.: Delving into adversarial attacks on deep policies. arXiv preprint arXiv:1705.06452 (2017)
23. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: Artificial intelligence safety and security, pp. 99–112. Chapman and Hall/CRC (2018)
24. Li, S., Zhao, B.Z.H., Yu, J., Xue, M., Kaafar, D., Zhu, H.: Invisible backdoor attacks against deep neural networks. arXiv preprint arXiv:1909.02742 (2019)
25. Li, Y., Jiang, Y., Li, Z., Xia, S.T.: Backdoor learning: A survey. IEEE Transactions on Neural Networks and Learning Systems (2022)
26. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks (2017)
27. Lovisotto, G., Turner, H., Sluganovic, I., Strohmeier, M., Martinovic, I.: {SLAP}: Improving physical adversarial examples with {Short-Lived} adversarial perturbations. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 1865–1882 (2021)
28. McGraw, G., Figueroa, H., Shepardson, V., Bonett, R.: An architectural risk analysis of machine learning systems: Toward more secure machine learning. Berryville Institute of Machine Learning, Clarke County, VA. Accessed on: Mar **23** (2020)
29. MITRE: CVE of Sockeye. https://www.cvedetails.com/cve/CVE-2021-43811/ (2022)
30. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1765–1773 (2017)
31. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2574–2582 (2016)
32. Morgulis, N., Kreines, A., Mendelowitz, S., Weisglass, Y.: Fooling a real car with adversarial traffic signs. arXiv preprint arXiv:1907.00374 (2019)
33. mozilla: DeepSpeech (Feb 2022), https://github.com/mozilla/DeepSpeech, [Online; accessed 8. Feb. 2022]
34. Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E.C., Roli, F.: Towards poisoning of deep learning algorithms with back-gradient optimization. In: Proceedings of the 10th ACM workshop on artificial intelligence and security. pp. 27–38 (2017)

35. Nassi, D., Ben-Netanel, R., Elovici, Y., Nassi, B.: Mobilbye: Attacking adas with camera spoofing (2019). https://doi.org/10.48550/ARXIV.1906.09765, https://arxiv.org/abs/1906.09765
36. Ning, R., Li, J., Xin, C., Wu, H.: Invisible poison: A blackbox clean label backdoor attack to deep neural networks. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications. pp. 1–10 (2021). https://doi.org/10.1109/INFOCOM42981.2021.9488902
37. Papernot, N.: A marauder's map of security and privacy in machine learning. arXiv:1811.01134 [cs] (Nov 2018), http://arxiv.org/abs/1811.01134, arXiv: 1811.01134
38. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia conference on computer and communications security. pp. 506–519 (2017)
39. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European symposium on security and privacy (EuroS&P). pp. 372–387. IEEE (2016)
40. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
41. Patel, N., Krishnamurthy, P., Garg, S., Khorrami, F.: Bait and switch: Online training data poisoning of autonomous driving systems (2020). https://doi.org/10.48550/ARXIV.2011.04065, https://arxiv.org/abs/2011.04065
42. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
43. Schuster, R., Shuster, T., Meri, Y., Shmatikov, V.: Humpty dumpty: Controlling word meanings via corpus poisoning. In: IEEE S&P (2020)
44. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.: Automated generation and analysis of attack graphs. In: Proceedings 2002 IEEE Symposium on Security and Privacy. pp. 273–284 (2002). https://doi.org/10.1109/SECPRI.2002.1004377
45. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE symposium on security and privacy (SP). pp. 3–18. IEEE (2017)
46. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Transactions on Evolutionary Computation **23**(5), 828–841 (2019)
47. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
48. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction {APIs}. In: 25th USENIX security symposium (USENIX Security 16). pp. 601–618 (2016)
49. Xiao, C., Li, B., Zhu, J.Y., He, W., Liu, M., Song, D.: Generating adversarial examples with adversarial networks. arXiv preprint arXiv:1801.02610 (2018)
50. Xiao, Q., Li, K., Zhang, D., Xu, W.: Security risks in deep learning implementations. In: 2018 IEEE Security and Privacy Workshops (SPW). pp. 123–128. IEEE (2018)

51. Xu, K., Liu, S., Chen, P.Y., Zhao, P., Lin, X.: Defending against backdoor attack on deep neural networks. arXiv preprint arXiv:2002.12162 (2020)
52. Yin, H., Molchanov, P., Alvarez, J.M., Li, Z., Mallya, A., Hoiem, D., Jha, N.K., Kautz, J.: Dreaming to distill: Data-free knowledge transfer via deepinversion. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8715–8724 (2020)
53. Yu, H., Yang, K., Zhang, T., Tsai, Y.Y., Ho, T.Y., Jin, Y.: Cloudleak: Large-scale deep learning models stealing through adversarial examples. In: NDSS (2020)
54. Yuan, X., Chen, Y., Zhao, Y., Long, Y., Liu, X., Chen, K., Zhang, S., Huang, H., Wang, X., Gunter, C.A.: {CommanderSong}: A systematic approach for practical adversarial voice recognition. In: 27th USENIX security symposium (USENIX security 18). pp. 49–64 (2018)
55. Zhang, L., Liu, P., Choi, Y., Chen, P.: Semantics-preserving reinforcement learning attack against graph neural networks for malware detection. IEEE Transactions on Dependable and Secure Computing (2022)
56. Zhao, Z., Dua, D., Singh, S.: Generating natural adversarial examples. arXiv preprint arXiv:1710.11342 (2017)
57. Zhou, L., Gao, J., Li, D., Shum, H.Y.: The design and implementation of xiaoice, an empathetic social chatbot. arXiv:1812.08989 [cs] (Sep 2019), http://arxiv.org/abs/1812.08989, arXiv: 1812.08989