A Threat Analysis Methodology for Security Requirements Elicitation in Machine Learning Based Systems

Carl Wilhjelm
Georgia State University
Atlanta, GA, USA
cwilhjelm1@student.gsu.edu

Awad A. Younis
Northern Kentucky University
Highland Heights, KY, USA
mussaa1@nku.edu

Abstract— **Machine learning (ML) models are now a key component for many applications. However, machine learning based systems (MLBSs), those systems that incorporate them, have proven vulnerable to various new attacks as a result. Currently, there exists no systematic process for eliciting security requirements for MLBSs that incorporates the identification of adversarial machine learning (AML) threats with those of a traditional non-MLBS. In this research study, we explore the applicability of traditional threat modeling and existing attack libraries in addressing MLBS security in the requirements phase. Using an example MLBS, we examined the applicability of 1) DFD and STRIDE in enumerating AML threats; 2) Microsoft SDL AI/ML Bug Bar in ranking the impact of the identified threats; and 3) the Microsoft AML attack library in eliciting threat mitigations to MLBSs. Such a method has the potential to assist team members, even with only domain specific knowledge, to collaboratively mitigate MLBS threats.**

Keywords— *Adversarial Machine Learning, Security Requirements Engineering, Requirements Elicitation Using Threat Modeling, STRIDE, Attack Libraries, Model Inference and Perturbation and Evasion Attacks.*

I. INTRODUCTION

ML models are now a key component for many applications. For example, in the field of medical diagnostics, MLBSs rely on advancements in deep learning for image processing. These systems, however, while vulnerable to the same threats that plague systems without ML components (we will refer to these as *traditional* systems and threats) have additionally been shown to be vulnerable to various types of AML threats (ex. poisoning attacks, model extraction attacks, and evasion attacks) that also have the potential for severe consequences [1], [2]. AML research studies the vulnerability of ML to the designs of a malicious actor [1]. A number of studies have investigated MLBSs' security, identified attacks and defenses, and proposed taxonomies for classification of these [1, 3, 4, 2, 5, 6, 7, 8]. Yet, AML researchers typically address their security concerns with a narrow focus on ML models, while a holistic paradigm to address both the AML threats and traditional security concerns of an MLBS concurrently remains almost wholly unexplored.

Recently, research studies [9], [10], [11] [12], [13], and [14] have investigated the applicability of traditional software engineering techniques to MLBSs and identified some challenges. Further studies have focused on investigating the importance and unique challenges of applying requirements engineering to MLBSs [15], [16]. A recent study [17] has also attempted to integrate security into an MLBS development life cycle (requirements analysis, design, implementation, testing, deployment, and maintenance) using a systematic application of knowledge, methodology, ML expertise, software engineering, and security. However, there exists no systematic process for eliciting security requirements for MLBSs that incorporates the identification of adversarial machine learning (AML) threats with those of a traditional non-MLBS.

In this research study, we explore the applicability of traditional threat modeling and existing attack libraries in addressing MLBS security early in system development particularly at the requirements phase. Our main objectives in this research are to 1) demonstrate the potential for identifying AML threats in MLBSs using data flow diagrams (DFDs) and STRIDE; 2) Rank the impact of MLBS threats using Microsoft AI/ML bug bar ranking method; and 3) Elicit AML threat mitigations using Microsoft AML attack library.

To identify threats related to MLBSs, the system under study needs to first be modeled. There exist several strategies to accomplish this including brainstorming, asset modeling, attacker modeling, and software modeling [18]. Of these, software modeling is the most focused on the software, which developers are expected to understand [18]. This contrasts with the others in not specifically requiring an understanding of the business, its assets, or the potential attackers which might otherwise require additional training. Several diagram schemes are available to model software. DFDs are among the most used in threat modeling due to their simplicity because they are considered lightweight and do not require any idiomatic knowledge of software engineering [18]. We will explore using DFDs to model an MLBS. Once the DFD model for the MLBS has been identified, we will focus on finding threats that are conceptually unique to MLBSs and explore the possible application of STRIDE.

STRIDE is a classification methodology for potential threats based on the threat categories of Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege [19]. STRIDE is among the most used approaches in threat modeling due to its maturity [20]. However, mapping STRIDE to MLBS threats is not entirely straightforward as AML threats are categorically different from their traditional security engineering counterparts. To overcome this challenge, we first propose exploring the AML literature [21], [22], and [23], to develop a taxonomy to categorize potential AML threats to a MLBS. We will then map

978-1-7281-8915-4/20/$31.00 ©2020 IEEE
DOI 10.1109/QRS-C51114.2020.00078

AML threats, at their highest level of abstraction, to the DFD and then to STRIDE, to bridge the conceptual gaps between STRIDE and AML threats. After that, we will explore using STRIDE on an example application to find the relevant AML threats to an MLBS. It is our expectation that this mapping will provide teams, possibly those composed of security engineers and data scientists without large overlaps in expertise, a shared understanding of the threats, and also how they might be mitigated. More details about our method can be found in section II.

Threat ranking techniques are used to prioritize which threats need to be mitigated first, if at all. There currently exist a number of threat ranking techniques among them the Common Vulnerability Scoring System (CVSS) [24], DREAD [25], and Bug Bars [18]. However, because the bug bar specifically is a very lightweight technique that is accessible to someone without domain specific knowledge, we will explore this option for our ranking technique. We will specifically investigate using Microsoft Bug Bar to rank AML threats related to MLBSs [22]. It ranks AML threat severity on a scale between critical, important, moderate, and low [26].

Attack Libraries are commonly used with threat modeling to identify specific threats and their mitigations in traditional security engineering [18]. They are constructed to track and organize threats. They have been found to increase developers' knowledge of the ways attackers work and especially those without expertise in cybersecurity [27], [28], [29], [30]. They can be either organization specific or use the ones published by the security community such as Common Attack Pattern Enumeration and Classification (CAPEC) [31]. CAPEC is a publicly available community-developed list of common attack patterns that provides comprehensive schema, classification taxonomy, and threat defenses or mitigations. As of the time this paper was written, CAPEC does not provide AML threat related attacks. However, we have found that Microsoft has built a publicly available attack library specifically for MLBS using AML literature [22]. This library describes how an adversary can attack a vulnerable MLBS and provides techniques to mitigate those threats.

The rest of the paper is organized as follows. Section II describes the proposed method based on threat modeling and attack libraries. Section III illustrates the proposed method with a case study. Section IV presents comments along with the issues that need further research.

II. PROPOSED METHOD

Our effort attempts to apply preexisting techniques in the field of software security engineering to elicit security requirements for MLBSs. We are particularly interested in exploring traditional threat modeling and existing attack libraries to elicit security requirements for AML threats in MLBSs. Threats and attack libraries help identify security requirements (a threat that is impossible to mitigate implies non- requirements) [18]. To provide an understanding of the security requirements elicitation for MLBSs, we provide an overview of a possible approach based on DFD and STRIDE, Microsoft AI/ML Bug Bar's threat ranking approach, and Microsoft's AML attack library. The overall view of our method is summarized in the following steps:

- Identify threats related to MLBSs using DFDs and STRIDE:
  - Develop a software model (an architectural model) for MLBSs using a DFD
  - Develop an AML threat taxonomy based on existing literature
  - Map the AML threat taxonomy to the DFD
  - Bridge the conceptual gaps between AML threats and STRIDE
  - Use STRIDE to identify AML threats related to MLBSs
- Rank AML threat impacts using Microsoft AI/ML Bug Bar's threat ranking approach.
- Elicit AML threat mitigations using Microsoft AI/ML attack library.

In the following subsections, we will illustrate our approach.

A. *HOW THREATS RELATED TO MLBSs COULD BE IDENTIFIED USING DFDs and STRIDE?*

The most important step in eliciting security requirements is the identification of potential threats. This could be accomplished using many strategies, for example, brainstorming, asset modeling, attacker modeling, or software modeling [18]. Of these, however, software modeling is the most focused on the software itself, which developers understand better than the business and its assets or the motivation and ability of possible attackers [18]. It describes how an adversary can attack a vulnerable MLBS and provides techniques to mitigate those

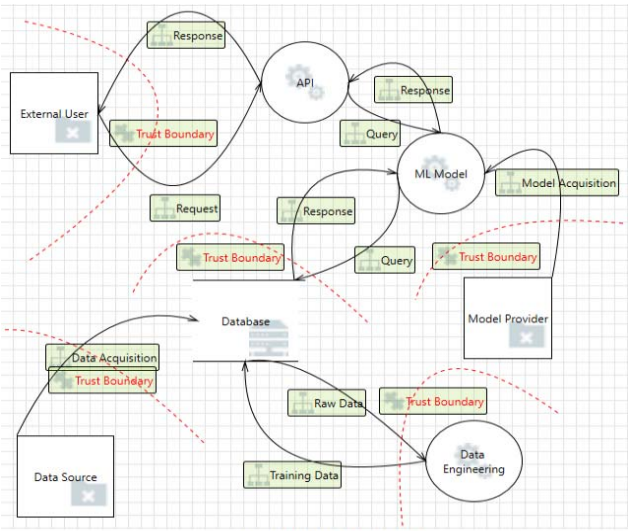

Fig. 1 Ex. MLBS Architecture.

threats. There are some diagram schemes used to better understand the software and its potential threats including DFDs and UML

1) DFD: DFDs are among the most used in threat modeling due to their simplicity [18]. They are lightweight and do not require any idiomatic knowledge of software engineering. In contrast to UML diagrams, which require experience in object oriented programming, we expect DFDs to make modeling MLBSs readily understood even by a pure data

scientist. To identify potential threats related to MLBSs, we will use a DFD to illustrate a possible architecture and should not expect any issues stemming from a lack of domain specific knowledge here. Fig. 1 shows an example.

2) STRIDE: Once a DFD model for MLBS has been identified, we will need a way to identify the threats implicit in it. STRIDE an acronym that stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege [19] [20]. This framework and mnemonic were designed to help software developers identify the types of attacks that traditional software tends to experience.

Here, we recognize that the specific threats articulated in the current literature on AML often do not immediately lend themselves to STRIDE or indeed to any traditional security requirements engineering techniques. Some have argued that this is because MLBSs are categorically different from their traditional counterparts [17]. MLBSs may, for example, contain sensitive information that if extracted could be used to execute other attacks and they are increasingly being used to implement security features e.g. facial recognition for biometric authentication. It is our conjecture that the complexity and specificity of the literature is the primary source of the idea that such an attempt is not likely to succeed.

With this in mind, we will begin by offering the following threat taxonomy based on the contributions of [21], [22], and [23], in order to abstract a schema for the categorization of AML threats, from which the applicability of traditional methods might be more clearly drawn. To this end, we generalize ML models into three components common to all functions: input, processing, and output. Respectively the AML threats can be classified based on their vector of attack, as follows:

- Flawed Data – A situation where bad data as a starting point, or deliberate provision of false data by a hostile party, results in one or more of the following:
  - Biased data or small sample size.
  - Incorrect, corrupted, or poisoned data.
- Model Extraction – Reverse-engineering the model from the system's behavior or outputs:
  - Through normal observation or access to a significant sample of sympathetic data through normal channels.
  - Through deliberately feeding specially tailored data to the model i.e. perturbation attack, without meaningful controls.
- Data Extraction – Threat actor obtains the training data used for the model from manipulating the system's behavior or inputs:
  - Sensitive data or personal data leaked.
  - Training data obtained, allowing reversing of model or inferences about algorithms.

We would expect each threat discussed above to be addressed across the following DFD trust boundaries:

- Flawed data - between the data source, database, and data engineering
- Model Extraction - between the external user and the API, as well as the external provider and the model.
- Data Extraction - between the database and external user via the ML model and API.

An approach to threat enumeration that might follow logically at this point would be to offer an addendum to STRIDE that simply includes the above threats (detailed justification for each point follows below), as seen in Table I.

TABLE I:
STRIDE PLUS ML-THREATS[a] IN ADDENDUM

DFD Element	S	T	R	I	D	E	FD	ME	DE
Data Flows		x		x	x				
Data Stores		x		x	x		x		
Processes	x	x	x	x	x	x	x	x	x
External Entity	x		x				x	x	

[a] S: Spoofing; T: Tampering; R: Repudiation; I: Information Disclosure; D: Denial of Service; E: Elevation of Privilege; FD: Flawed Data; ME: Model Extraction; DE: Data Extraction

However, this approach has two problems built into it: 1) it has a great deal of redundancy, and 2) it continues to rely on AML domain specific knowledge for threat enumeration. We will show that using STRIDE to directly enumerate MLBS threats, is adequate to express the same points. Thus, we will use the above taxonomy to bridge the conceptual gaps between STRIDE and AML threats. Table II shows how STRIDE will be mapped to the AML threat taxonomy above.

Mapping Flawed Data (FD) to STRIDE

- Spoofing (External Entity) - In MLBSs, the threat of spoofing is associated with those systems being used for security, for example, if an ML model used for facial recognition is compromised as a result of its inadequate training data.
- Tampering (Data Store / Process) - While data poisoning is malicious by definition and on its face an example of tampering, data bias and small data size, are often subjective and sometimes unrecognized problems. Yet, if we are willing to consider that these latter issues also violate the property of integrity, i.e. the data is "modified" relative to its ideal state, it seems plausible to the authors that mapping even these misuse cases to tampering may be workable, even absent malicious intent whether inherent in data or the process of data engineering. While it might make sense to further expand this concern to data flows, wherein data might be tampered with in transit for the purposes of data poisoning, we will resign these concerns to the traditional domain.
- Repudiation (Process / External Entity) - Similarly with the issue of spoofing, AML threats can make repudiation difficult if the models are relied on for example for identity authentication.

- Information Disclosure - Data engineering techniques that fail to adequately address privacy may be the source of information disclosures, particularly internally. These may further propagate up to data being extracted from the model. However, the issues related to information disclosure from data itself are already addressed by traditional threat modeling.
- Denial of Service (Data Store / Process) - Considered in depth, any misclassification might lead to a denial of service, for example if someone should have been approved for something but was denied based on data bias.
- Elevation of Privilege (Process) - Particularly in a security related system, elevation of privilege may occur as a result of an information disclosure or authentication failure. It is also possible that data poisoning could compromise the entire model.

Mapping Model Extraction (ME) to STRIDE

- Spoofing (Process / External Entity) - Spoofing attacks may lead to the extraction of data necessary for these attacks, for example faking credentials to get the output of an ML model through its API, however this is an issue already being covered by traditional concerns. More importantly, successful model extraction may ultimately lead to further spoofing attacks in security related MLBS.
- Tampering (Process) - Possibly the most interesting category to consider in model extraction in a sense similar to the one described under flawed data. Frequently, the ultimate goal of model extraction attacks is the ability to predict the output of a model such that an input can be crafted to obtain it. Yet, the input itself is not being tampered with per se and nor is the model. Thus, we consider this as tampering with the ideal output of the model, i.e. an evasion attack.
- Repudiation (Process) - It is again conceivable that in a security related MLBS this might be a concern, as understanding the model may be the first step toward an attack on the mechanism of authentication.
- Information Disclosure (Process) - The most straightforward threat, information disclosure can describe how the training data is obtained for the attacker's model.
- Denial of Service (Process) - While examples in the literature are not widespread, it is conceivable that a model input may if not properly handled result in a denial of service attack. For example, unconstrained size where input is expected to be a certain size.
- Elevation of Privilege (Process) - When trusted for authentication, elevation of privilege is an obvious analogy to model extraction. The potential for models to be compromised to the point of being unusable in addition to unhandled errors makes this category also worth considering.

Mapping Data Extraction (DE) to STRIDE

- Spoofing - While this may be involved in executing the attack, or a consequence if sensitive information is extracted from such an attack, it is not necessary to consider specifically under our definition of data extraction.
- Tampering - Tampering as discussed under model extraction may be involved in this attack but it is not its object.
- Repudiation - We may consider repudiation again in the traditional sense.
- Information Disclosure (Processes) - This is the ultimate goal of a data extraction attack. The vulnerability that makes the attack possible exists between the database and the external user. Data engineering is only one possible solution depending on the model being used.
- Denial of Service - There does not seem to exist any literature suggesting this as a possible category under these attacks.
- Elevation of Privilege - While it is possible for an elevation of privilege attack to occur as a result of data extraction where sensitive information used for authentication is exposed, this is more accurately addressed under information disclosure.

TABLE II:
MAPPING STRIDE TO ML-THREATS[b]

STRIDE-Threats\ ML-Threats	S	T	R	I	D	E
Flawed Data	EE	DS/P	P/EE		DS/P	P
Model Extraction	P/EE	P	P	P	P	P
Data Extraction				P		

[b] DF: Data Flow; DS: Data Store; P: Process; EE: External Entity

In Table 3, we show all threats relevant to MLBSs: Traditional, Flawed Data, Model Extraction, and Data Extraction under their STRIDE categorizations (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege). Thus, providing a unified expression of all potential threats to an MLBS.

TABLE III
ML-THREATS[c] IN STRIDE

Element	S	T	R	I	D	E
Data Flows		T		T	T	
Data Stores		FD		T	FD	
Processes	ME	FD/ME	FD/ME	ME/DE	FD/ME	FD/ME
External Entity	FD/ME		FD			

[c] FD: Flawed Data; ME: Model Extraction; DE: Data Extraction. T: Traditional Threats

B. HOW THE IMPACT OF AML THREATS COULD BE RANKED?

After threats have been identified, which threat to address first can be decided by using established threat ranking techniques such as DREAD [25], CVSS [24], or a bug bar. Ultimately, there is a high degree of subjectivity in the consideration of severity and individuals or organizations may consider rank them differently. As the field of AML is still evolving, we propose using a bug bar as the ranking technique and leave examining DREAD, CVSS or other ranking techniques to future work. In bug bar, bugs are given a severity based on a shared understanding of their impact. In this paper, we will use directly the rankings from the Microsoft AI/ML Bug Bar to rank AML threats to our system, which is available online [22].

C. HOW AML THREAT MITIGATIONS COULD BE IDENTIFIED

In traditional security engineering, attack libraries are commonly used with threat modeling to increase a developer's knowledge of possible attacks and provide ways to address them [18]. Attack libraries are constructed to track and organize threats and are either self-developed or published by the security community, i.e. Common Attack Pattern Enumeration and Classification (CAPEC) [31]. CAPEC is a publicly available community-developed list of common attack patterns that provides a comprehensive dictionary and classification taxonomy for threats and describes defenses or mitigations for them, but it does not currently address AML related threats.

Microsoft has built a publicly available attack library specifically for MLBS which describes how an adversary may attack a vulnerable MLBS and provides techniques to tackle these threats using available AML literature [20]. Critically, however, this attack library requires specialized knowledge in ML and does not lend itself to understanding by traditional security requirements engineers. Although the available attack libraries for MLBS still need improvement, in our case study we found using Microsoft's resources sufficient to identify significant threats and mitigation techniques. We will only include select details therefrom, as they are quite extensive, but will refer back to their source for further reading when appropriate.

III. EVALUATION AND RESULTS

Our approach in eliciting security requirements for MLBSs is based on the sequential steps discussed in section II. To implement our approach, we will first introduce a selected case study, then demonstrate the method and explore some partial results. Further exploration of its effectiveness will be discussed in section IV.

A. Case Study

Currently, machine learning is being applied to two areas of particular importance to the insurance industry, loss prediction and fraud detection. Loss prediction seeks to predict the amount an insurance company will lose on a given policy, e.g. if the insured driver is a young male, they are statistically more likely to incur a loss. Fraud detection typically tries to determine if a specific claim is fraudulent, e.g. if a claim occurs within a month of the policy being opened, this is a red flag for fraud and indeed is highly correlated to real fraud. The latter is where we will ground our case study.

Insurance companies often operate through brokers, individuals or companies that may offer insurance policies from multiple companies to potential clients, for a commission. At times, these same brokers are responsible for filing claims with the company and dispensing funds to customers. In fact, depending on the arrangement a great deal of responsibility is put on the broker for the successful operation of the insurance industry [32]. Thus, it is conceivable that a broker might be privy to enough data to infer models that the insurance company is using, making it vulnerable to the following implementation of a model inference attack. Such systems currently proliferate the industry in uncounted variations and are potentially subject to such attacks [33].

Consider a hypothetical fraud detection model that takes a claim, submitted by a broker through a company's API, and provides the company with a likelihood of fraud on a scale from 0-9 inclusive. If the likelihood is less than 10%, the company automatically approves the claim with no human being ever looking at it. Otherwise, the company takes some days to process or investigate the claim relative to the likelihood of fraud. Then as soon as any investigations into the claim are completed, the API responds immediately to the broker. Given a moderately sophisticated actor, it is conceivable that they would be able to deduce that a significant subset of claims were being processed automatically, hereafter referred to as zero day approvals.

The goal then of the attacker is to produce a scheme whereby fraudulent claims could be crafted in such a way as to garner a zero day approval, and to do this the attacker would attempt to recreate the model from the output of its API. In this context, a Model Inference Attack, if successful, would allow a malicious broker to test possible permutations of claims against their simulated model such that they could confidently submit a fraudulent claim without a human being ever looking at it. Thereby executing a successful Evasion Attack.

Our simulation will seek to emulate a modern insurance company which has developed a machine learning model based on their own internal data, that outputs a probability of fraud in a given claim, and initiates an investigation process which takes a proportionately related number of days to investigate said claim. Inspired by a blog using the same data, to predict fraud which can be found at [34]. It seems to take its data from a GitHub repository the original source of which could not immediately be found [35]. In an actual attack, the malicious broker's own database of API submissions and response times would provide the data to train an inferred model with which they could confidently launch an evasion attack, providing zero day approvals to fraudulent claims.

1) Source Data and Engineered Data

The target of our original data was a binary value for fraud detected. It contained columns such as: months as customer, age, insured sex, insured education level, insured hobbies, incident type, number of vehicles involved, property damage, bodily injuries, witnesses, police report available, total claim amount, injury claim, and property claim. While it had the serious disadvantage of only having 1000 elements, due to a

lack of preferable alternatives, we moved forward with this dataset, as it was possible to engineer into an acceptable scheme.

In the case of logistic regression, fraud will be predicted in any instance where the likelihood of fraud is greater than the likelihood of not fraud. After creating a model to predict fraud that had a mean accuracy of 80%, we took the output of the probability of fraud for each element and parsed that into a range from 0-9 inclusive. While this simulation does not directly address whether or not payment was issued, these values replaced the fraud reported column entirely under a new column, days in processing nullsec, meaning the ground truth for days processed, with no threat mitigation.

2) The Attack: Extracting the Model

Given the hypothetical situation as outlined above, we can assume that any attacker would have a meager output sample relative to the size of the training data. However, due to the limitations of our dataset it was necessary not only to reuse the same rows that were used to train the model but to use a statistically significant subset of those, in our case 10%. We also assumed for the sake of the black-box simulation that the features that would be used for the actual model would not be known to the attacker and therefore removed different columns from consideration.

The object of this model, however, is not to distinguish between varying levels of likelihood of a claim being fraudulent, but to evade the detection of fraud. So, from this subset of the data we further narrowed the data used in the attacker's model down to only those rows representing the zero day approvals and the label 1 approvals, representing fraud predicted somewhere between 10% and 19%. The idea being that this would provide even greater precision to the attacker in being able to determine whether or not the given entry would trigger human handling at any level.

We then set up another logistic regression, as this is a common approach to such problems, and trained it on our binary scheme. To evaluate the effectiveness, we then tested this model on a binary evaluation for every row in the original data, with the result being a model that is 96% accurate. Of course, it is logical to assume that this result is largely due to the limitations of the data, but it is nonetheless representative of what we might expect to find in this scenario.

Thus, with a small sample of output, and some basic machine learning knowledge, the attacker is able to create a model with which to test possible perturbations of claims such that even if fraudulent they will garner a zero day approval. Thereby successfully implementing a model inference attack, and opening the door to successful evasion attacks.

B. Implementation

As described in subsection A of section III, the attack vector in our simulation is entirely dependent on the output of the insurance company's API, the willingness or desire of the insurance company to output the results of its fraud investigations as soon as they are completed, and we assume the system is otherwise secure. Therefore, we will illustrate our method's ability to identify the threat at this point in the system and employ the mitigation strategy proposed in the Microsoft Bug Bar. We will first need to model the system.

1) Identify AML threats to MLBSs using DFD and STRIDE

In practice, our methodology calls for a collaboration of teammates working on potentially complex systems and identifying AML threats concurrently with traditional ones. For demonstration purposes we will simplify this process and make some assumptions to focus on the results as they concern AML threats to our simulation specifically. While production level systems will certainly require a case specific DFD we will use the example presented in Fig. 1 with the exception that we assume that both the model and data are internally created. We therefore will ignore the data source, data acquisition, and model source elements. Further, our ML model is not a security component and we therefore will ignore threats to external entities (as justified in Subsection A of Section II), as we are concerning ourselves only with AML threats. We will also ignore the possibility of biased data, as the limitations our training data prohibits this possibility.

As in subsection A of section II under DFD, we consider the trust boundaries associated with our data flows, data stores, processes, and external entities. We extract their AML threats from the STRIDE classifications in Table III based on element type, consider the relevant threats based on our own architecture (in bold), and reference the attacks we categorized from the Microsoft Bug Bar in subsection B of section II. The identified threats for the fraud detection system are shown in table IV.

TABLE IV:
IDENTIFIED ML-THREATS[d] FOR THE FRAUD DETECTION SYSTEM.

Threat ID	Threat[c]	Element	S	T	R	I	D	E
T1	Data Poisoning	Database		**FD**			FD	
T2	Data Poisoning	Data Eng.	ME	FD/ ME	FD/ ME	ME/ DE	**FD**/ ME	FD/ ME
T3	Membership Inference	Data Eng.	ME	FD/ ME	FD/ ME	ME/ **DE**	FD/ ME	FD/ ME
T4	Adversarial Perturbation	API	ME	FD/ ME	FD/ ME	ME/ DE	FD/ ME	FD/ **ME**
T5	Model Stealing	API	ME	FD/ ME	FD/ ME	ME/ DE	FD/ ME	FD/ **ME**
T6	Membership Inference	API	ME	FD/ ME	FD/ ME	ME/ **DE**	FD/ ME	FD/ ME

[d]FD: Flawed Data; ME: Model Extraction; DE: Data Extraction, [c]See Microsoft Bug Bar [26]

2) Rank MLBS threat impacts using Microsoft AI/ML Bug Bar threats ranking approach.

From here, we will focus on threats to the API to limit the scope of our discussion. We can ignore membership inference

attacks because our training data contains no sensitive information. After parsing Adversarial Perturbation into its specific threats: Targeted Misclassification, Source/Target Misclassification, Random Misclassification, and Confidence Reduction, we refer to the severity of these threats as provided by the Microsoft Bug Bar, shown in Table V.

TABLE V:

LIST OF THREATS AND THEIR MICROSOFT AI/ML BUG BAR SEVERITY VALUE.

Threat ID	Threat[e]	Severity[f]
T4A	Targeted Misclassification	Critical
T4B	Source/Target Misclassification	Critical
T4C	Random Misclassification	Important
T4D	Confidence Reduction	Important
T5	Model Stealing	Important

[f]See Microsoft Bug Bar [26]

3) Elicit MLBS threats mitigations using Microsoft AML Attack Library.

We further narrow our focus to "targeted misclassification" and "model stealing" which we rank as critical and important respectively. It is important to note here specifically that in our attack we are in effect stealing the model through our Model Inference Attack to inform our targeted misclassification i.e. our Evasion Attack, and therefore we will only require a mitigation of the latter to succeed in the former. Because we are working in the absence of sufficiently accessible attack libraries, which according to our method should list these attacks under Elevation of Privilege, we will work from the AML threat classification to our threat enumeration, where otherwise we could approach from the direction of its STRIDE classification. We reference the mitigation from the Microsoft Bug Bar listed under SR3 in Table VI, an appropriate corroboration of the threat of information disclosure [22].

TABLE VI:
LIST OF ELICITED SECURITY REQUIREMENTS FOR EVERY THREAT.

Threat ID	Security Requirements ID	MLBS Security Requirements[g]
T4A	SR1	Highly Confident Near Neighbor (HCNN), a framework that combines confidence information and nearest neighbor search, to reinforce adversarial robustness of a base model.
T4A	SR2	Adversarial inputs are not robust in attribution space. Masking a few features with high attribution leads to change indecision of the machine learning model on the adversarial examples whereas natural inputs are robust in attribution space.
T5	SR3	Minimize or obfuscate the details returned in prediction APIs while still maintaining their usefulness to "honest" applications.

[g]See Microsoft Bug Bar [26]

For our mitigation we will employ SR3 through an API Hardening technique wherein the true output of the model is obfuscated to prevent the attacker from recreating it. In our case it is useful to consider the length of the investigations. The inclination of the company is to provide the result of an investigation as soon as it is completed. As such, the API's response time corresponds proportionally with the confidence of the model that fraud is taking place.

In our implementation we considered targets representing increments of 10% confidence. We can hypothesize therefore that if this system were changed to represent every 5% interval of confidence, i.e. a scale of 0 to 19, the ability to infer the model would be even greater on the part of the attacker. Intuitively, the opposite is also true. Clustering intervals of confidence in groups of maximum 50% likelihood (exclusive) of fraud, and minimum 50% likelihood of fraud, would provide the least confidence on the part of the attacker (that is still testable). This is the basis for the mitigation we will propose.

Given an API that groups the processing scales of 0-4 and 5-9 together, e.g. even a zero day approval would not be processed, through the API, until a delay equal to that of a minimum 40% likelihood, nor a five on the scale approved prior to a delay equal to that of a minimum 90% likelihood, the ability of the attacker to infer the model would logically be crippled, and we can confirm this result in our simulation.

To simulate the attack on the hardened API we will again train our data on the subset of 100 elements. However, this time we will not further parse our training data as we now have a clustered classification system, one label representing the 0-4 scale approvals which will all be released after the same delay, and another representing the 5-9 scale approvals which will all be released after the same delay. Note that we also cannot further reduce our training set as before as we have no way of evaluating the distinction between the model's zero day approvals and minimum 10% likelihood processing times.

When we calculate the effectiveness of this model in guessing the zero day approvals based only on this binary approach we see that it is only 40% accurate in doing so, and even more considerable, when we attempt to filter out only the top 10% most likely zero day approvals using this approach, we find an average success rate of 12%. Thus, granting the limitations of the simulation, demonstrating a complete mitigation of this attack vector.

Of course, the benefit of being able to control the situation entirely through the fact of its being a simulation leaves much to be desired in terms of applicability. Unfortunately, while this mitigation is demonstrably effective, in practice it may not be feasible for practical business reasons or otherwise. Nonetheless, in evaluating the concept of API Hardening it is a useful example for how consideration for the information disclosure potential of maximizing the utility of the API might have adverse consequences for the security of the system.

IV. CONCLUSION AND FUTURE WORK

As MLBSs proliferate and evolve the consequences of ignoring threats against them will become increasingly severe. In this paper, we addressed two areas of concern on this subject 1) the lack of a well established systematic approach to eliciting security requirements for MLBS and 2) the inaccessibility of the available literature to traditional security requirements engineers. The complex nature of developing security requirements for MLBSs was revealed and one possible

technique for developing security requirements for MLBSs was articulated, evaluated, and discussed.

The proposed method was illustrated using a single case study, the efficacy of which needs to be further evaluated. To this end, further work would not only test further different more complex case studies, but field-test the proposed method to see how well the techniques provide coverage in threat models of real-world systems. We also plan to assess the usability of our method by introducing it to students in one or more of the following courses: machine learning, big data programming, software engineering, or software security engineering at the undergraduate and graduate levels in our organization.

We adapted the Microsoft Attack Library and Bug Bar in this research, other attack libraries or self-developed ones, however, could also be used to widen the coverage of threats. We ultimately plan to develop our attack library using AML literature and to experiment with applying CVSS or DREAD to assess the severity of the MLBS threats more objectively. The method discussed in this research is mostly a manual process, but designing and implementing a tool that automates part of the process to make it easier for data scientists and security engineers to apply the proposed method in their MLBS development we believe would be a more practical solution worth working towards.

REFERENCES

[1] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar, "Adversarial machine learning," Proceedings of the 4th ACM workshop on Security and artificial intelligence - AISec '11. 2011.

[2] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and A. T. Ristenpart, "Stealing machine learning models via prediction apis," presented at the In 25th Security Symposium ({USENIX} Security 16), Austin, TX, 2016, pp. 601–618.

[3] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," Communications of the ACM, vol. 61, no. 7. pp. 56–66, 2018.

[4] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "SoK: Security and Privacy in Machine Learning," 2018 IEEE European Symposium on Security and Privacy (EuroS&P). 2018.

[5] F. Khalid, M. A. Hanif, S. Rehman, and M. Shafique, "Security for Machine Learning-Based Systems: Attacks and Challenges During Training and Inference," 2018 International Conference on Frontiers of Information Technology (FIT). 2018.

[6] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. C. M. Leung, "A Survey on Security Threats and Defensive Techniques of Machine Learning: A Data Driven View," IEEE Access, vol. 6. pp. 12103–12117, 2018.

[7] E. Tabassi, K. J. Burns, M. Hadjimichael, A. D. Molina-Markham, and J. T. Sexton, "A taxonomy and terminology of adversarial machine learning." 2019.

[8] M. Ozdag, "Adversarial Attacks and Defenses Against Deep Neural Networks: A Survey," Procedia Computer Science, vol. 140. pp. 152–161, 2018.

[9] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison, "Investigating statistical machine learning as a tool for software development," Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08. 2008.

[10] S. Amershi et al., "Software Engineering for Machine Learning: A Case Study," 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 2019.

[11] F. Ishikawa and N. Yoshioka, "How Do Engineers Perceive Difficulties in Engineering of Machine-Learning Systems? - Questionnaire Survey," 2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP). 2019.

[12] Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does Machine Learning Change Software Development Practices?," IEEE Transactions on Software Engineering. pp. 1–1, 2019.

[13] C. E. Otero and A. Peter, "Research Directions for Engineering Big Data Analytics Software," IEEE Intelligent Systems, vol. 30, no. 1. pp. 13–19, 2015.

[14] N. H. Madhavji, A. Miranskyy, and K. Kontogiannis, "Big Picture of Big Data Software Engineering: With Example Research Challenges," 2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering. 2015.

[15] J. Horkoff, "Non-Functional Requirements for Machine Learning: Challenges and New Directions," 2019 IEEE 27th International Requirements Engineering Conference (RE). 2019.

[16] V. Andreas and M. Borg, "Requirements Engineering for Machine Learning: Perspectives from Data Scientists," 13-Aug-2019.

[17] Y. Liu, L. Ma, and J. Zhao, "Secure Deep Learning Engineering: A Road Towards Quality Assurance of Intelligent Systems," Formal Methods and Software Engineering. pp. 3–15, 2019.

[18] A. Shostack, Threat Modeling: Designing for Security. John Wiley & Sons, 2014.

[19] F. Swiderski and W. Snyder, Threat Modeling. Microsoft Press., 2004.

[20] N. Shevchenko, T. A. Chick, P. O'Riordan, T. P. Scanlon, and C. Woody, "THREAT MODELING: A SUMMARY OF AVAILABLE METHODS," SOFTWARE ENGINEERING INSTITUTE | CARNEGIE MELLON UNIVERSITY, Jul. 2018.

[21] N. Dunn, "Building safer machine learning systems – A Threat Model." 28-Aug-2018.

[22] A. Marshall, J. Parikh, E. Kiciman, and And Ram Shankar, "Threat Modeling AI/ML Systems and Dependencies," Security documentation, 10-Nov-2019. [Online]. Available: https://docs.microsoft.com/en-us/security/threat-modeling-aiml#aiml-specific-threats-and-their-mitigations. [Accessed: 17-Dec-2019].

[23] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial Attacks and Defences: A Survey," Sep-2018.

[24] P. Mell, K. Scarfone, and S. Romanosky, "The common vulnerability scoring system (CVSS) and its applicability to federal agency systems." 2007.

[25] M. Howard and D. LeBlanc, Writing Secure Code. Pearson Education, 2003.

[26] "SDL Security Bug Bar (Sample)," Docs, Security, Security Development Lifecycle, 02-Dec-2018. [Online]. Available: https://docs.microsoft.com/en-us/security/sdl/security-bug-bar-sample#Definition_of_Terms]. [Accessed: 17-Dec-2019].

[27] M. N. A. Mohammad, M. Nazir, and K. Mustafa, "A Systematic Review and Analytical Evaluation of Security Requirements Engineering Approaches," Arabian Journal for Science and Engineering, vol. 44, no. 11. pp. 8963–8987, 2019.

[28] M. N. Johnstone, "Modelling misuse cases as a means of capturing security requirements," presented at the In 9th Australian Information Security Management Conference, 2011, p. 140.

[29] M. T. J. Ansari, D. Pandey, and M. Alenezi, "STORE: Security Threat Oriented Requirements Engineering Methodology," Journal of King Saud University - Computer and Information Sciences. 2018.

[30] X. Yuan, E. B. Nuakoh, I. Williams, and H. Yu, "Developing Abuse Cases Based on Threat Modeling and Attack Patterns," Journal of Software, vol. 10, no. 4. pp. 491–498, 2015.

[31] "CAPEC: Common Attack Pattern Enumeration and Classification," 2007-2019. [Online]. Available: retrieved from http://capec.mitre.org/. [Accessed: 17-Dec-2019].

[32] B. Downs, "What Does An Insurance Broker Do? - Business Benefits Group," Business Benefits Group, 07-Nov-2016. [Online]. Available: https://www.bbgbroker.com/what-does-an-insurance-broker-do/. [Accessed: 21-Dec-2019].

[33] "Using data analytics, AI technology to curb benefits fraud." [Online]. Available: https://www.benefitscanada.com/news/using-data-analytics-ai-technology-to-curb-benefits-fraud-120628. [Accessed: 21-Dec-2019].

[34] A. Dommalapati, "Using a Data Pipeline to Predict Insurance Claim Fraud," Medium, 05-Jun-2019. [Online]. Available: https://blog.usejournal.com/using-a-data-pipeline-to-predict-insurance-claim-fraud-6f8e85367294. [Accessed: 21-Dec-2019].

[35] "jodb - Overview," GitHub. [Online]. Available: https://github.com/jodb. [Accessed: 21-Dec-2011