The Impact of Platform Vulnerabilities in AI Systems

by

Ashley Kim

B.S. Electrical Engineering and Computer Science,
Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Author . . . .
Department of Electrical Engineering and Computer Science
August 14, 2020

Certified by . . . .
Howard Shrobe
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by . . . .
Hamed Okhravi
Senior Staff, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

The Impact of Platform Vulnerabilities in AI Systems

by

Ashley Kim

Submitted to the Department of Electrical Engineering and Computer Science
on August 14, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Artificial intelligence has become increasingly prevalant through the past five years, even resulting in a national strategy for artificial intelligence. With such widespread usage, it is critical that we understand the threats to AI security. Historically, research on security in AI systems has focused on vulnerabilities in the training algorithm (e.g., adversarial machine learning), or vulnerabilities in the training process (e.g., data poisoning attacks). However, there has not been much research on how vulnerabilities in the platform on which the AI system runs can impact the classification results. In this work, we study the impact of platform vulnerabilities on AI systems. We divide the work into two major part: a concrete proof-of-concept attack to prove the feasibility and impact of platform attack, and a higher-level qualitative analysis to reason about the impact of large vulnerability classes on AI systems. We demonstrate an attack on the Microsoft Cognitive Toolkit which results in targeted misclassification, leveraging a memory safety vulnerability in a third party library. Furthermore, we provide a general classification of system vulnerabilities and their impacts on AI systems specifically.

Thesis Supervisor: Howard Shrobe
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Hamed Okhravi
Title: Senior Staff, MIT Lincoln Laboratory

Acknowledgments

Thank you to Hamed Okhravi for guidance through the entirety of the process. I could not have accomplished any of the work here without your ideas and your help.

Thank you to Howie Shrobe and Hamed Okhravi for giving me the opportunity to work with the group, and providing the idea for this project.

Thank you to Kyle Denney for the AI and reverse-engineering specific advice through our meetings, and for organizing information on various threats to AI systems. The work provided formed a large basis for the threat overview.

Contents

List of Figures

List of Tables

List of Code Listings

Chapter 1

Introduction

Artificial intelligence has become increasingly prevalent in the past few years, being used to solve problems across domains of speech recognition [89][4], optimization [80][73], and object identification [77][2][28]. AI has also been adopted in many sensitive and fault intolerant fields such as fraud detection [76] and medicine [30]. With the rise in open source toolkits, it has become easier than ever to deploy AI in a wide variety of systems, causing their use to be even more widespread.

ML as a subdomain of AI has received special attention recently. ML systems are often opaque, however [20]. The models are trained over many iterations without needing much interaction with the underlying system. This can make vulnerabilities in the system much more difficult to analyze. The validity of the results obtained from machine learning relies on three main properties: sufficient training data, robustness of the algorithm against adversarial samples, and the security of the platform on which the engine runs. While both the properties of robustness in training [36][9][1][27] and the algorithm [57][14][88][90][26] have been studied extensively, the third property, the security of the platform, has received little attention. Unlike in many other systems, in an AI engine, a single change in a variable may result in changes that are drastic and difficult to diagnose. Thus, the impact of platform vulnerabilities on AI systems have to be studied to understand their implications for real-world scenarios.

In this thesis, we study the impact of threats to AI platforms. We first build a proof-of-concept attack to show that platform attacks are not only feasible, they can have impacts that have received very little attention in the literature. For instance, a malicious sample can maliciously modify not only its own classification but also the classification of other samples. We demonstrate a cross-sample attack to highlight the importance of AI platform security. Second, we study the larger landscape of attacks against AI platforms, and analyze their impact against AI engines. In particular, we find that certain classes of attacks that are important in enterprise settings are much less important for AI systems, while others that seem weak in enterprise settings, such as limited data-only attacks, have major impact on AI systems.

Our work demonstrates a novel class of cross-sample attacks that can impact AI systems, and raise awareness towards the less-considered results of attacks against the AI system platform. We also discuss a more general threat landscape that highlights important threats against AI engines. In doing so, we identify gaps in current techniques used in securing AI systems, which the research community can leverage in order to develop newer techniques.

There has been substantial reseach in securing AI platforms. However, a large majority of them focus on either attacking the algorithm used in the learning process [7][10][88][59][26][90][14] or the training process [36][9][1][27]. There has been some recent work in designing a secure platform for AI systems, but a majority focus on securing the platform in a data-privacy setting, attempting to limit the amount of private data exposed to an ML system during the training [33][64][37][54]. There has been some recent work in guaranteeing the integrity of AI systems in the cloud using trusted execution environments (TEEs) such as Intel SGX [43] [81], but these systems are designed with threats in the untrusted software stack in mind, not necessarily vulnerabilities in the AI system itself. In contrast to these efforts, we present a specific attack against AI systems that only leverages vulnerabilities in the AI system itself, and demonstrate a need for addressing platform security in a AI-specific setting.

In this thesis, we demonstrate a specific cross-sample attack against a popular machine learning framework (Section 3), and proceed to discuss general threats against AI systems (Section 4). We then discuss the other efforts in the field of AI security (Section 5), and provide a discussion of the results (Section 6), identifying vulnerability classes that are not well addressed by currently existing research.

Chapter 2

Background

To contextualize our research, we provide a background for AI as a whole, and in AI security.

2.1 AI Systems

AI has become increasingly prevalent in software in the past few years [29][13]. Many critical systems, such as autonomous vehicles [39][24] and robots [12][91], rely on AI for correct functionality.

Machine learning (ML), a subset of AI techniques, generally automates the process of analyzing data sets, producing models that classify inputs based on the correlations it finds in the training data it is initially provided [3]. There are three main classes of machine learning techniques, depending on what data is provided as initial input during the training phase. The three types are supervised learning, unsupervised learning, and reinforcement learning [57]. Supervised learning refers to methods that initially provide the systems with sets of inputs and their corresponding outputs. Unsupervised learning initially provides inputs without information about the expected outputs. Reinforcement learning relies on providing sequences of actions and observations as inputs, with the goal of identifying the optimal actions to take in a specified environment.

Although separately classified by the inputs to the system, AI systems generally

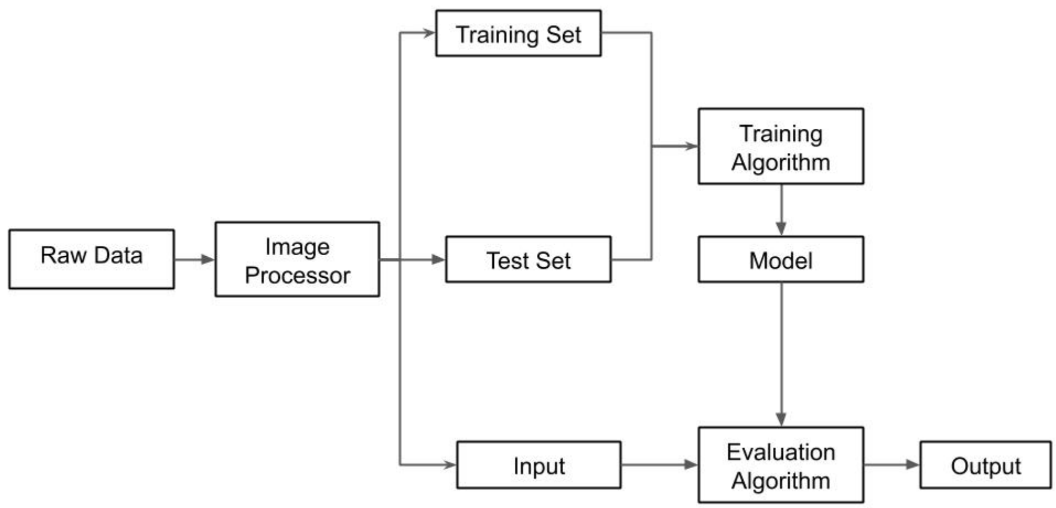

Figure 2-1: Information flow through a general AI System

have similar attack surfaces, because their data processing pipelines are similar. A generic system, illustrated in Figure 2-1, takes an object or action from the physical domain, and translates it into input through various methods (sensors, cameras, etc.). Then, the digital representation of the data is sent through the machine learning model, which then outputs a result [57].

Tampering of the data at any point in this process could cause incorrect outputs of the system. If the inputs are tampered with either in the physical domain or any time during the conversion to digital data, the outputs may be unexpected. In addition, if there are errors in the model itself, the outcomes may also be incorrect.

The validity of the model in artificial intelligence rely on three main factors: good training, robustness of the algorithm, and security of the platform. All three elements are necessary to be able to trust the results of the classifications.

2.2 Problem Definition

AI systems have risen in popularity recently, and there has been an increased interest in the security of AI systems. Various researchers have worked on demonstrating that machine learning is vulnerable to attacks through the training data [36][9] or through adversarial examples that leverage flaws in the algorithms [14][57][90]. However, as we stated, there are three main facets of machine learning. We need robustness in our training, our algorithms, and in the platform on which the system operates. While the first two facets have been researched [57], there has been little work (to our knowledge) on the last facet.

There is an underlying system that generates and trains the networks, and runs the inputs through the classifier. The vulnerabilities in the underlying systems could thus result in incorrect results even if the algorithms and the training data were all free of errors. As AI systems are used to make decisions for sensitive applications in self-driving cars, drones, or robots [14], trust in their behavior is imperative. Thus, there is a need to learn about how platform vulnerabilities can effect AI systems.

There has been a large amount of general research done in systems security [78][17][44][75][32], but none focuses on machine learning systems specifically. While general systems vulnerabilities can still apply to machine learning platforms, we hypothesize that there are specific classes of vulnerabilities that are significantly more critical on AI systems than anywhere else. For example, an attacker being able to overwrite a single data value may not be so damaging in general, but may be critical in an AI system, where a single change in a model parameter is hard to identify, and can change the results drastically.

Thus, we will attempt to find and classify vulnerabilities in AI platforms, with a strong focus on ones that behave differently in AI platforms. Mostly, these should be data-only vulnerabilities, and we will seek to demonstrate that their impact is much greater than previously thought. Thus, we will motivate a specialized threat model for AI systems, and identify methods for securing these platforms against threats.

Chapter 3

Real-World Attack

In this section we demonstrate an exploit on a existing AI system. Our exploit targets Microsoft CNTK, a open-source deep learning library. We leverage a memory safety vulnerability in a third party library in order to cause targeted misclassification. The exploit relies on the existence of the vulnerability specified in CVE-2018-5268.

3.1 Microsoft Cognitive Toolkit

The Cognitive Toolkit (CNTK) is a open-source, deep learning toolkit developed by Microsoft [68]. Primarily, it allows the user to define a network as a directed graph, and train and evaluate the specified model. As illustrated in Figure 3-1, it takes in a model specification along with inputs that are passed through a image processor, and outputs a model that it can then evaluate with arbitrary processed inputs. As it allows fine-grained model specification, CNTK can express nearly arbitrary neural networks, and execute the corresponding algorithms. CNTK can be loaded in as a library in Python, C#, or C++, but for our purposes, we use it as a standalone tool and explicitly define our models using BrainScript, their custom model description language.

The BrainScript interface allows us to specify one of two modes, training or evaluation, for execution. In the training mode, the BrainScript file specifies the model layout and the size and shape of all parameters. In the evaluation mode, CNTK just

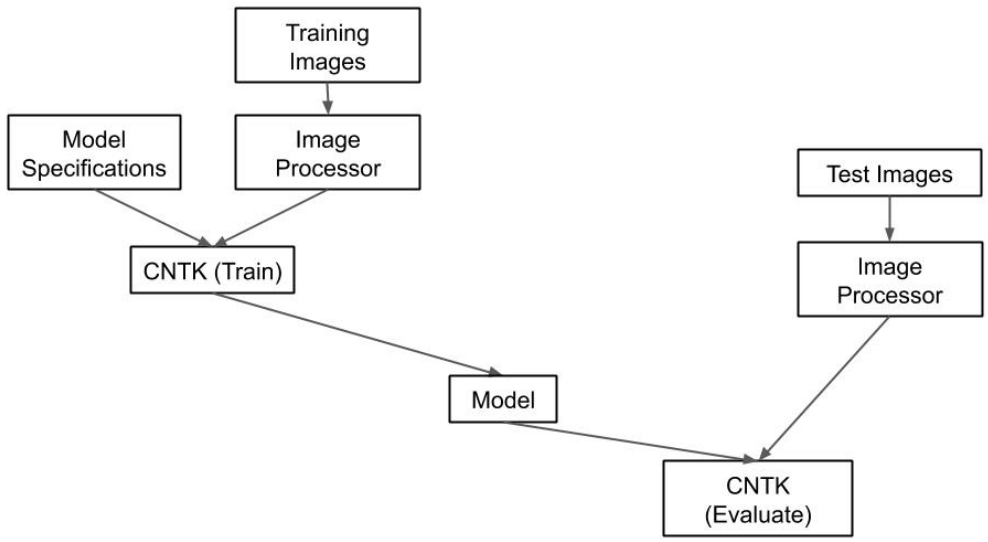

Figure 3-1: Overview of CNTK

needs a path to the trained model and the input images and their classifications.

When running CNTK providing the BrainScript specification as an input, CNTK parses the specifications and initializes the model, located on the heap. All components of the model are allocated on the heap, and pointers are initialized according to the model description. The matrices holding the input data for the model is initialized to empty during this stage, and references to it are created even though it's not yet populated. Figure 3-2 illustrates the structure of the matrix containing the input data. It contains maps corresponding to features and labels for the potential input set, which each contain a map with a pointer `m_pArray` that contains a reference to the input data corresponding to the matrix.

The inputs are loaded through a call to `ImageReader`, which fills a buffer on the stack with image data read from a specified text file. This buffer contains a matrix, and also holds all the information about the layout and size of the input. The layout and size information is included in the model itself, and the process will abort if an input that is invalid for the current model is provided. The matrix contains pointers to four different types of matrices. There are sparse and normal versions of matrices for both CPU and GPU based calculations. These submatrices are what holds the

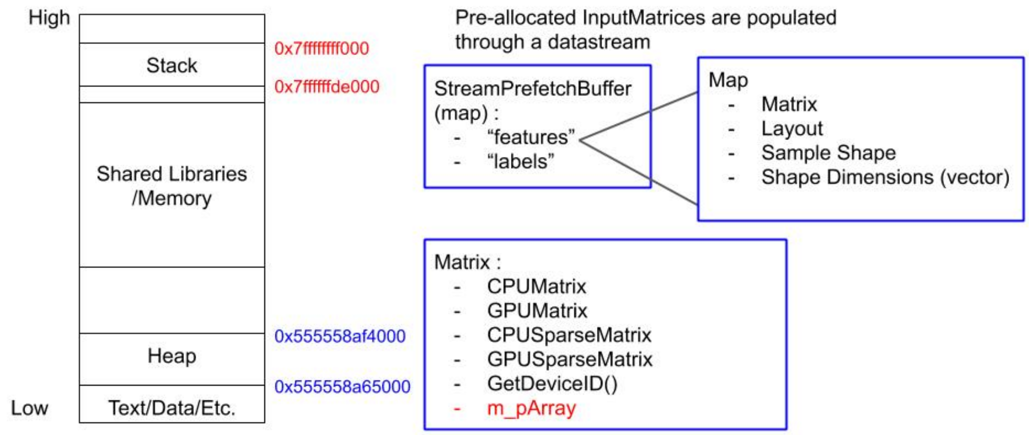

Figure 3-2: The structure of the buffer storing loaded images, and the pointers it contains.

pointer to the numerical data stored in the matrix. However, the data is accessed through getter and setter methods in the matrix class.

After the image data is read into the buffer, CNTK swaps it into the input matrix that was previously linked into the model. As the references to the matrix already exist in the network, they swap the contents of the buffer and the matrix instead of switching pointers.

3.2 Example Exploit

The goal of our attack is to use a malicious input image for CNTK and cause the engine to misclassify other images. This requires finding a memory corruption vulnerability in either CNTK or the third party libraries it loads, and leveraging it during the classification process.

We studied the image loading process for possible corruptible sites. We found that the images were converted into matrix form in `Readers/ImageReader/ImageReader.cpp`, which made calls out to `Readers/ImageReader/ImageDataDeserializer.cpp` in order to parse the image data. In line 98 of `Source/Readers/`

ImageReader/ImageDataDeserializer.cpp, shown in listing 3.2.1, they call the OpenCV library to read the image from the path.

Listing 3.2.1: Call to OpenCV

```
return cv::imread(path, grayscale ?
        cv::IMREAD_GRAYSCALE : cv::IMREAD_COLOR);
```

OpenCV is a popular open-source computer vision library, popular for machine learning applications due to its focus on efficiency and support for various types of specialized hardware [11]. OpenCV had a heap-based buffer overflow that could be triggered by parsing a malicious image in the loading process (CVE-2018-5268). We decided to leverage this vulnerability to introduce a write-what-where vulnerability [78] in CNTK itself.

Given a write-what-where vulnerability, we then determine which addresses to overwrite in order to cause a misclassification. We observe that the following steps occur during the image loading process.

1) The input matrix is initialized, and references to it are created within the network.

2) The image data is loaded into a buffer.

3) The matrix and the buffer have their contents swapped.

Step 3) of the process specifically swap the contents of the two matrices, instead of changing their pointers. This is due to the initialization in step 1), causing references to the input matrix to be remembered. This means that the image data referenced by the input matrix will be the one used during classification. This is key to our attack, and the specific timeline is outlined in Figure 3-3.

We considered several approaches before deciding on a plan of attack. We first considered attempting to modify the size of the image in some way, in an attempt to misalign the evaluation frame with the input images. Robust bounds checking and layout verification made this approach infeasible. Then, we considered overwriting the data pointer in the input matrix used in the network. However, after our window

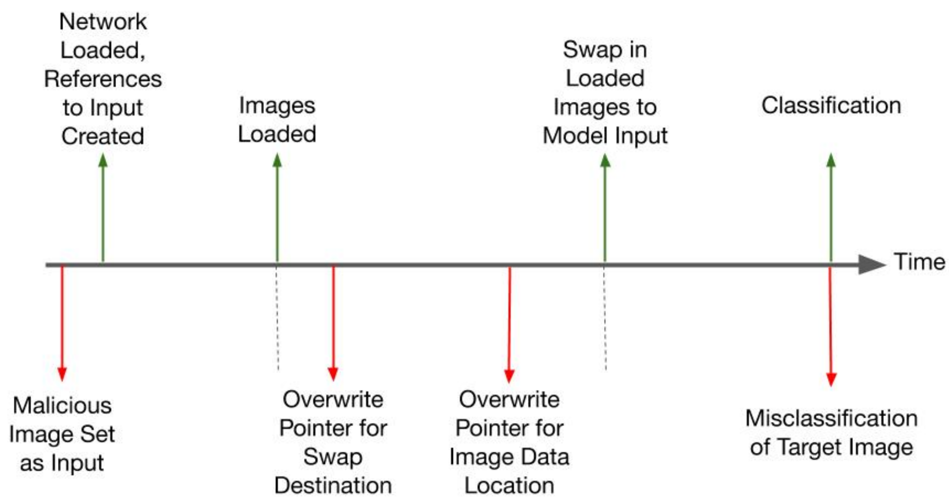

Figure 3-3: The timeline of events in the classification process, with adversary actions outlined in red. The dotted lines indicate the window of opportunity for the adversary to leverage the arbitrary write vulnerability.

for arbitrary writes has passed, we have a swap that overwrites the value of the data pointer with that of the input. We cannot delay our write, as we are leveraging a vulnerability in an external library that is called exactly once during the image loading process in step 2).

Thus, we need to accomplish two things with our vulnerability. We need to ensure the swap does not overwrite the desired data in our input matrix, and overwrite the data pointer in the input matrix to produce the desired classification.

We accomplish the first goal by creating a dummy matrix object, and overwriting the destination of the swap. Specifically, the swap occurs as follows in lines 301-307 of `CNTK/Source/Readers/ReaderLib/ReaderShim.cpp::301-307`, shown in listing 3.2.2.

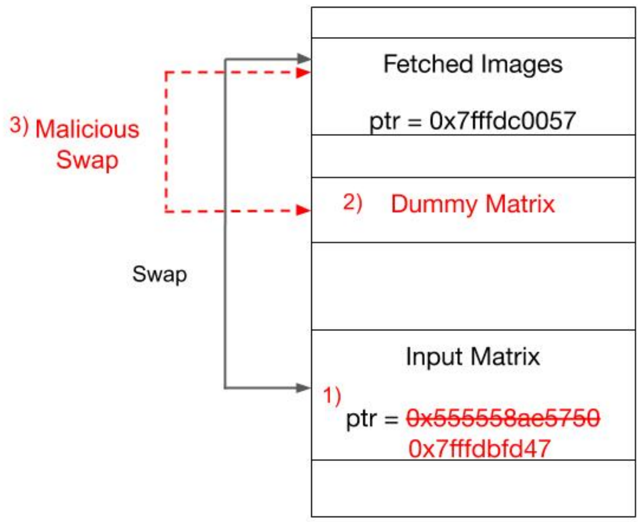

Figure 3-4: The sequence of the adversary's actions and their effects on the memory of the system.

Listing 3.2.2: Matrix Swap Snippet

```
for (auto i = matrices.begin(); i != matrices.end(); ++i)
{
    std::swap(i->second.GetMatrix<ElemType>(),
          *m_prefetchBuffers[i->first].m_matrix);

      // Resetting layouts.
    i->second.pMBLayout->Init(1, 0);
}
```

Before the swap, we can overwrite the matrices object, ensuring that the entries it holds are not the ones referenced in the network. Then, this swap will effectively result in a no-op. Thus, if we overwrote the data pointer in the input matrix, the change will be preserved and the evaluation will occur on the malicious input data. The sequence of transformations is illustrated in Figure 3-4.

Now, we need to overwrite the data pointer in the input matrix. If the buffer loaded the image data into memory location `X`, we considered two possibilities for a corrupted pointer: `X - image_size`, and `X + image_size`. The first possibility can corrupt the first image of the stream, and in a targeted fashion if we leverage additional writes during the corruption step in order to load a fake image right before the real images. However, this approach is inconsistent, as the region of memory right before the image buffer is not guaranteed to be writeable. The second approach is much less brittle, but has the disadvantage that we can only corrupt a classification into one of an image that was truly in the input stream.

We chose the second approach because the attack already assumes the attacker can upload a malicious input image to the classifier. Thus, the disadvantage is mostly mitigated, and we get a much more reliable attack.

As a result of uploading the malicious image, the attack executes the series of commands described in Listing 3.2.3.

Listing 3.2.3: Exploit Pseudocode

```
// corrupt the matrix data pointer
target_ptr = matrices.begin()->second.GetMatrix<Float>().m_CPUMatrix.get()
            .m_sob.get()).m_pArray
fetched_images_ptr = *m_prefechBuffers[i->first].m_matrix..m_CPUMatrix.get())
            .m_sob.get().m_pArray
target_ptr = test_images_ptr + 4 * 28 * 28

// undo the swap
dummy_matrix_ptr = make_fake_matrix_on_stack();
matrices.begin() = dummy_matrix_ptr
```

This overwrites both the swap destination pointer and the image data pointer, causing the system to receive essentially an offset batch of inputs. Several examples of the corruption are depicted in Figure 3-5.

We ran this attack on version 2.7 of CNTK, leveraging the vulnerability from version 3.4.0 of `OpenCV`. We discovered that we could corrupt the classification of all input images to the classification of the next image. This caused the evaluation accuracy to fall from 100% to 0%. Using a sample of the two images shown, the results of the outputs varied as shown in Figure 3-5. Even though we only provided

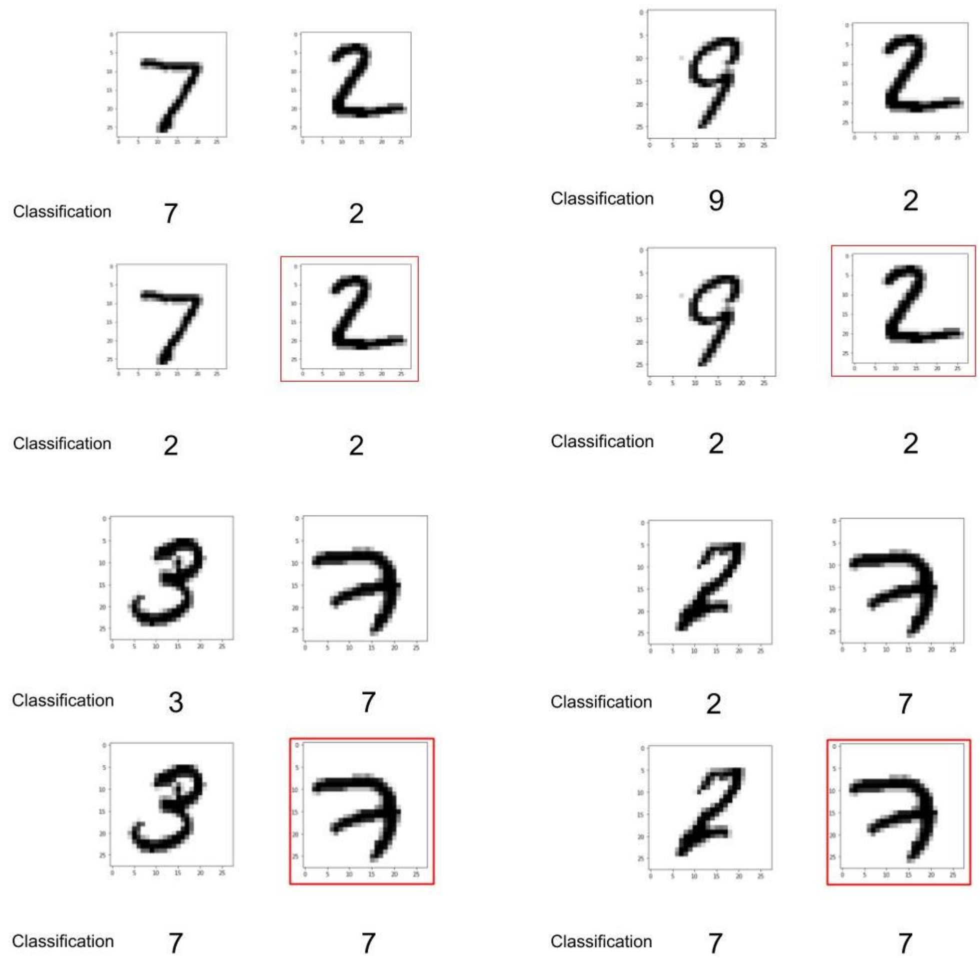

Figure 3-5: Impact of the exploit. Without a malicious image, the classifications are correct, but with a malicious exploit the classifications of all images are impacted.

one malicious image to the process, we managed to alter the classification of all other images.

Chapter 4

Broader Threat Analysis

We provide an overview of general attacks on AI systems, how they might be executed, and their impacts on the system. AI Attacks can be classified intro three large classes: platform attacks, which attack the platform the AI system runs on, algorithm attacks, which attack the algorithm the AI system uses for classification, and data attacks, which insert data into the system to cause malfunction. These attacks can further be subdivided into more specific categories. Platform attacks can be divided into data modification attacks, which modify the data in the evaluation step of the system to cause misclassification, denial of service attacks, which causes a failure of availability in the system, and input leakage attacks, which gain access to confidential data that passes through the platform. Algorithm attacks are described by adversarial examples, samples that are specifically constructed to cause a misclassification by the classification algorithm. Data attacks can also be divided into training data poisoning attacks, which insert malicious inputs to the training set to cause misclassification, or training data leakage attacks, which use malicious inputs to identify samples from the confidential training set. This categorization is illustrated in Figure 4-1.

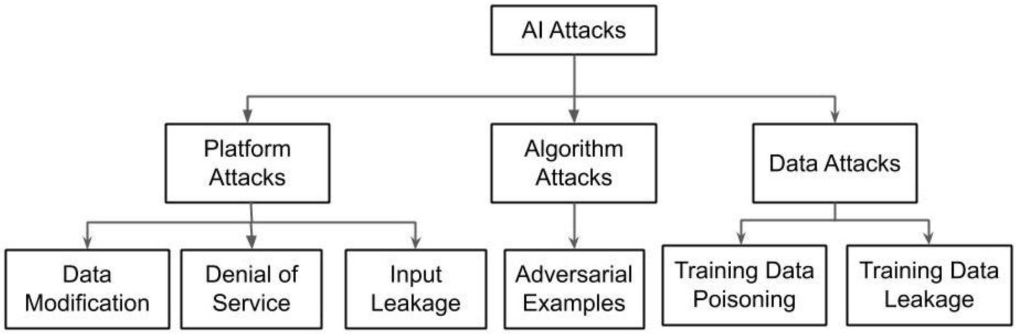

Figure 4-1: Tree of AI Security

4.1 Platform Attacks

4.1.1 Data Modification Attacks

This section focuses on attacks that modify the data in the input evaluation step of the AI system. This can range from modifying the parameters in the model, to modifying the inputs to the system maliciously. These attacks do not involve changes to the control flow of the system.

Attack Method

These attacks require that the adversary be able to overwrite a parameter to the system. Depending on how objects are stored in memory, there are various methods [19][63][35] an attacker could use in order to compromise a system by overwriting parameters.

Most AI systems take in an input, which is then categorized by the system. Consider an example machine learning system that interprets its input as described in listing 4.1.1.

Listing 4.1.1: Vulnerable Stack Overflow Code

```c
modelObject model;
char input[200];

load_model(model);
gets(input);

return evaluate_model(model, inputs);
```

This snippet loads in a model, reads an input, evaluates the input based on the model, and returns the result. However, the key observation is that the model and input are both stored as local variables in memory on the stack. As this model uses `gets()`, which does not verify the size of the input, when loading in the model, the adversary could provide an input of size much larger than the program specification, which in this case is 200. If this occurs, `gets()` will attempt to copy the entire adversary-provided input into the buffer, overflowing the data into the rest of the local variables. As the model is stored on the stack in the same contiguous section as all the other local variables, this buffer overflow could modify the model parameters to attacker-controlled values and corrupt the evaluation of the input. While this example overwrites memory on the stack, we can also perform a very similar attack to overwrite parameters on the heap, as described in listing 4.1.2.

Listing 4.1.2: Vulnerable Heap Overflow Code

```c
char *input = malloc(20);
modelObject *model = malloc(modelSize);
strcpy(input, argv[1]);

return evaluate_model(model, inputs);
```

Since `strcpy()` reads to the nullbyte in `argv[1]`, we may write more than 20 characters into the input buffer, causing an overflow. As the model is allocated right after the input, the adversary can overwrite model parameters at will with the attack. Furthermore, the adversary is not necessarily limited to attacking data that is directly adjacent to the input in memory. Consider the system described in listing 4.1.3

Listing 4.1.3: Vulnerable Arbitrary Overwrite Code

```c
char *input = malloc(20);
inputIndex = 10;
modelObject *model = malloc(modelSize);
strcpy(input, argv[1]);

model[inputIndex] = input;

return evaluate_model(model, inputs);
```

The snippet illustrates a function where the AI system evaluates the model given an input with state, and returns the element at the index corresponding to the output. However, the `inputIndex` is adjacent to the input in memory, and the input uses a vulnerable function that allows a buffer overflow. This allows the attacker to control the value of `inputIndex`. As C arrays do not natively check array boundaries, the attacker can assign any value to `inputIndex`, and thus write to a value at any desired offset from the model. This allows the adversary to have arbitrary write access over the system.

The previous attacks take advantage of vulnerable system functions, for example `gets()` and `strcpy()`, in the code. However, this is not necessarily required for the adversary to be able to corrupt the data on the system. Many modern classifiers attempt to support various types of inputs, and rely on third-party libraries in order to read these values. In the example we provided in the thesis, we leveraged a buffer overflow vulnerability in the library used to load the image, managing to overwrite other images in the sample. Using this approach, we managed to corrupt not only the classification of the adversary-provided data, but of entirely benign data provided in the same sample set.

Attack Impact

Attacks that modify the weights of the model itself can lead to a variety of effects. Even uncontrolled overwriting of a few parameters can cause the model to behave in unintended ways, returning random classifications instead of the desired ones. Given more control, if an attacker has the ability to overwrite specific parameters with attacker-controlled values, they can achieve targeted misclassification and insert a backdoor. However, as the model parameters were modified, we have consistency in our results. Inputs will be classified the same (possibly incorrect) way every time it is provided as an input to the system. This raises the possibility that, by causing reproducible errors to the system, the adversary would be detected.

Attacks that modify single inputs can have a much more subtle impact on the system. As demonstrated in our example exploit, if a malicious entity could provide an input that modified other inputs to the system, we can achieve targeted misclassification on specific non-malicious inputs. As no changes are being made to the model, this attack is difficult to detect, and when the benign image is provided to the system in a sample without the malicious image, it will be classified correctly, causing the error to be irreproducible in a testing environment.

Defenses

Defenses against data modification attacks require strict guarantees of data integrity in the entire system. There has been research in technology that enables complete access controls on specific sections of memory, such as XFI [21] that guarantees dataflow integrity. We could also mitigate these attacks somewhat by having randomization at the data-level [61]. Data-space randomization is a technique that randomizes the representation of data in memory, instead of its location. The code encrypts and decrypts the data on every access, but by using multiple keys and not storing raw data in memory, we can mitigate many data only attacks [8].

More sophisticated defenses involve enclaving technology, such as Intel SGX [16]. These systems allow the users to separate sections of code and data in a system into a trusted core, which provides additional integrity guarantees to the system. For example, the operators could put the core of the evaluation engine along with the model data into a secure enclave, isolating it from all the third party libraries loaded into the system. Nested enclaves were demonstrated to be useful for isolating software systems from their third party libraries [60] and this usage could be extended to AI systems. However, in recent years, SGX has been shown to be vulnerable to various side-channel leakage attacks, thus new enclaving primitives are needed in the community to enable such protections [86][56].

4.1.2 Denial of Service

This section focuses on attacks that cause the AI engine to crash, or slow itself and other processes.

Attack Method

A denial of service attack can be performed in one of several ways. If the engine is hosted on the internet, the attacker may "flood" a network, sending a large amount of traffic and denying access to legitimate users [34]. This could be executed by targeting the network specifically, executing an amplification attack [72][5] or a UDP flood attack [45] [41].

However, the user could also utilize the specific attributes of AI engines. Classification of a single sample with a large network could take several seconds to minutes each. Each classification may be a relatively expensive process in terms of computing resources, so if an adversary gains the ability to provide inputs to the system at a fast rate, the process could be bottlenecked and be unable to return results to legitimate users. Even if this method is mitigated by limiting the number of requests per user, a user may submit a few very large files for classification, stalling the system.

Attack Impact

A denial of service attack causes a failure of availability in the system. In addition to users not being able to access the system, if there are any other systems that rely on the AI engine, the system would be unable to function. If the systems were entirely reliant on the AI engine, even if all the training data was still available, it may be impossible to recover the results that are necessary for the system [52]. Having said that, denial of service attacks are often easy to detect and are not very stealthy.

Defenses

There are many existing defenses for network based denial of service attacks, such as monitoring or traffic filtering [6]. For AI engine specific denial of service attacks, we may limit both the maximum input size/complexity and the number of requests from a single source.

4.1.3 Input Leakage

There are also data privacy concerns with inputs that are provided to the ML system. If the system becomes compromised, it is possible for the adversary to gain access to all user inputs.

Attack Method

There are several methods the attacker could utilize to gain access to inputs that were provided to the ML system.

Firstly, if the model is hosted on the cloud, if the cloud service provider is malicious, they can gain access to all data stored on and sent through the AI system trivially.

More interestingly, the attacker can leverage platform vulnerabilities that allow them to gain read access to any of the data stored on the system [69][31][8]. For instance, consider a ML system as described in listing 4.1.4:

Listing 4.1.4: Arbitrary Read

```
char *input = malloc(20);
outputIndex = batchNum;
modelObject *model = malloc(modelSize);
strcpy(input, argv[1]);

outputs =  evaluate_model(model, inputs);
return outputs[outputIndex];
```

As in the data modification attack, the attacker has a buffer overflow in the input. This allows them to control the `outputIndex` parameter. As the system returns the result that is indexed by an adversary-controlled value, the adversary can print arbitrary chunks of data in the system.

Furthermore, there are various side channel attacks that can leak information about the system [67][40]. For instance, the adversary may be able to deduce details about the model based on the amount of time it takes to classify an attacker-provided image. While timing attacks are a threat to many systems [67][18], they are especially dangerous against AI systems, where a single classification can take a long time, amplifying the timing differences in the system.

In addition to timing attacks, AI systems are vulnerable to side channel vulnerabilities in the hardware they rely on. Recently, there was a major side channel attack, Spectre [40], that leveraged speculative execution in processors to leak information about running programs. AI systems would be vulnerable to Spectre and similar attacks on the microprocessor, and we need guarantees of secure hardware to prevent such side channel attacks.

Attack Impact

In either type of attack, the adversary fully compromises the confidentiality of the system. If the entire platform for the AI system is adversary-controlled, the adversary has access to data on the entire model, the inputs, and the training data provided to the system.

Attacks that leverage a specific platform vulnerability to gain arbitrary read over the system can also potentially read all parameters of the model, along with any inputs in the batch. However, if the network does not explicitly retain information about the inputs, the attacker cannot learn about any input submitted to the system in a different batch.

Defenses

Researchers have posited that homomorphic encryption provides a good defense to this problem [33]. Homomorphic encryption allows the system to run evaluations on encrypted data, and return an output that corresponds to the encrypted version of the output that corresponds to the original data [25]. This allows the system to run computations or train a model on user-provided input without gaining any information about the input itself. This manages to mitigate both the problem of a curious server or a malicious adversary attempting to leak information through the system. However, homomorphic encryption is way too slow to be of practical usage in modern AL systems and significant more research to mature this area.

4.2 Algorithm Robustness Attacks

4.2.1 Adversarial Samples

This section discusses attacks that utilize a vulnerability in the algorithm used by the AI system in order to cause a misclassification.

Attack Method

The best studied technique is the one of adversarial examples [58][47][14], where the adversary leverages weaknesses in the classification algorithm in order to misclassify an image. Often, this involves the adversary submitting a malicious image, for example, a stop sign, with non-human discernable noise added that causes the classifier to label it incorrectly, for example, as a green light [14].

Attack Impact

This technique can effectively cause the AI engine to misclassify any adversary-inputted image. However, as the image itself is being tampered with to cause the misclassification, the engine will behave correctly on all non-malicious images.

Defenses

There has been a large amount of research in detecting adversarial samples in a system [22][53][74]. We can mitigate adversarial examples to an extent by using adversarial training, using a training set with adversarial examples included [70]. We can also leverage the fact that, since adversarial samples are formed by specifically modifying images based on the specific model, adding small noise in the model is much more likely to change its classification, using that to distinguish possibly adversarial samples [85].

4.3 Data Attacks

This section focuses on attacks where the adversary uses data inputted to the AI system, either during the training or the evaluation phase, that causes the system to malfunction.

4.3.1 Training Data Poisoning

In this section, we focus on attacks that involve inserting malicious data into the training set of an AI system.

Attack Method

Training data poisoning involves inserting carefully crafted data into the training set with the intent of entering specific attack points into the data [9]. These attacks take advantage of the linear regression that underlies most AI systems that currently exist

[87]. The poisoning attack can be modeled as solving an optimization problem as follows [36].

$$\begin{aligned} &\arg\max_{D_p} \quad W(D', \theta_p)\,, \\ &\text{s.t.} \qquad\quad \theta_p \in \arg\min_{\theta} L(D_{tr} \cup D_p, \theta)\,. \end{aligned}$$

In the equation, W represents a loss function on a validation data set D'. The attacker seeks to find a value of D_p, the poisoned data set, that maximizes this loss. The constraint represents retraining the model on the training set D_{tr} augmented by the training set D_p. The loss function W varies based on the type of attack being attempted. If the attacker desires specific targeted misclassification, the loss function should evaluate only those specific inputs. If the attacker wants indiscriminate misclassification, the loss function should evaluate all possible inputs [36].

Attack Impact

There are two main types of poisoning attacks, poisoning availability attacks and poisoning integrity attacks [36], which have different impacts on the system. Poisoning availability attacks cause the model to malfunction on random inputs, effectively resulting in a denial of service on the system. Poisoning integrity attacks only affect a small subset of results, designed by the adversary, while preserving the integrity of other sample classifications.

Defenses

There are two main types of defenses against data poisoning, noise-resilient regression algorithms and adversarially resilient algorithms.

Noise-resilient regression algorithms rely on removing outliers from a dataset [65][83]. For example, if the adversary added poisoning data that was widely different from the normal distribution of the training data, they would be removed. However, this does not prevent the adversary from generating poisoning data that is distributed similarly to the original data [36]. As AI systems are sensitive to small perturbations, the adversary can still gain a large amount of control over the system.

Adversarially resilient algorithms rely on strong guarantees on the data and noise distribution [15][49]. These methods use the guarantees to prove robustness about the output data. However, we often do not have such strict guarantees on real-world data, rendering the technique impractical.

4.3.2 Training Data Leakage

Attack Method

Two major attacks that cause data leakage are model inversion attacks and membership inference attacks [62][23][71]. Model inversion attacks attempt to rebuild the features that were used when constructing the model. These reconstructed features reflect the average value of a classification set [23]. In settings such as facial recognition where an average sample in a class can reflect a person's identity clearly, this poses great risks for data privacy.

Membership inference attacks are slightly different in that they attempt to directly predict whether a sample was in the training data used for the model [66][62][71]. In models that are trained using confidential data, such as medical history, membership in the training set itself may be confidential [71].

Attack Impact

Training leakage attacks compromise the privacy of the large amounts of data used during the training process. Model inversion attacks can reconstruct approximations of the training data, and membership inference attacks can result in being able to explicitly identify samples that were used during training. Furthermore, if the system retains information about all data it classified and became compromised, the private data of all users that utilized the AI engine would be exposed to the adversary.

Defenses

Model inversion attacks require precise confidence values to function well, and it was found that reporting rounded confidence values greatly decreased the feasibility of this attack [23].

Membership inference attacks rely on knowing the relative confidence values of the model on possible labels for the input. One way to defend against this attack is to return only the class label without any of the confidence values, but even this is not entirely effective [71].

4.4 Summary

We summarize our findings in Table 4.1. We classify Arbitrary write, adversarial examples, and poisoning integrity attacks as high severity, due to their combination of high stealth and their fine-grained control over the failures in integrity they cause. Although denial of service attacks can impact all samples, it was not considered a high severity attack due to its low stealth. Different classes of data modification attacks, although having similar reproducibility, impact, and stealth, were separated in severity by how likely they were to accomplish a specific targeted corruption. A linear stack overflow, for instance, is much less likely to be able to influence the result of a specific sample than an arbitrary write. We deemed input leakage a fairly low risk, as they cannot modify the integrity of the system, and can be mitigated through encryption techniques, albeit with added overhead. Poisoning availability attacks have mostly similar impacts to denial service attacks, so we categorized them in similar severity. Training data leakage attacks, while still causing a failure of confidentiality with high stealth, were deemed only medium severity, as they require a large number of requests to the system to function, which can be additionally mitigated by protection against denial of service attacks.

Category	Subcategory	Attack	Severity	Reproducibility	Stealthiness	Samples Affected
Platform Attacks	Data Modification	Stack Buffer Overflow	Low	Low	High	Targeted Subset
		Heap Buffer Overflow	Medium	Low	High	Targeted Subset
		Arbitrary Write	High	Low	High	Targeted Subset
	Input Leakage	Input Leakage	Low	High	High	Some/All
	Denial of Service	Amplification Attack	Medium	High	Low	All
		UDP Flood	Medium	High	Low	All
Algorithm	Adversarial Examples	Adversarial Examples	High	High	High	Single
Data Attacks	Training Data Poisoning	Poisoning Availability Attack	Medium	High	Low	All
		Poisoning Integrity Attack	High	High	High	Targeted Subset
	Training Data Leakage	Model Inversion	Medium	High	High	All
		Membership Inference	Medium	High	High	All

Table 4.1: Categorization of Attacks on AI Systems

Chapter 5

Discussion

In our research, we have uncovered several gaps in existing research for AI security.

First, we need much more research into designing secure platforms for AI systems. We need to provide guarantees to the system that provide memory safety guarantees on the learning process itself, and resilience to vulnerabilities in third party libraries it requires. These guarantees would remove the memory and platform vulnerabilities from possible security problems, and disable the cross-sample attacks of the type illustrated in this thesis. There has been research in using TEEs to separate sensitive parts of the stack from the untrusted parts, but they rely on technology such as SGX, which is known to be vulnerable to side channel attacks [86]. Furthermore, many AI systems rely on GPUs for accelerating the training process, and support in enclaving techniques should extend to the different hardware.

In addition to building enclaving technology, it is imperative that we reduce the amount of potentially vulnerable dependencies for an AI system. Avenues for memory safety vulnerabilities that are leveraged through a third party application will be significantly reduced if we migrate them to fully memory safe languages. This will reduce the attack surface for a AI platform significantly. However, as this requires a significant and very costly development effort, new approaches are needed to 'sandbox' linked libraries and isolate their potential compromises from the AI engine, itself.

In addition, there is a demand for performant memory encryption throughout the system. Although there has been work in fully homomorphic encryption during both the training and evaluation stages of a AI system, the performance overheads make it prohibitive in practice for a commonly used system [79]. Thus, for guarantees on privacy for the input data, we need memory encryption through the system.

In addition, the research in adversarial examples is still lacking. There are a large number of techniques for combating adversarial examples, including input reconstruction [42], network verification [38], network distillation [58], and adversarial retraining [82]. However, these defenses each cover a fairly narrow range of possible attacks, and the reason for the existence of adversarial samples is still unknown [90]. This causes the design of robust proactive defenses to become difficult.

Chapter 6

Related Work

Many researchers have studied security in AI systems, and we discuss the attempts to exploit and secure AI systems in various ways. However, we must note that there is little, if not none, of research being done on the specific impact of platform vulnerabilities on AI systems. Thus, platform-based security analysis of our type is necessary to provide a complete view of the robustness of AI systems.

6.1 Security in the Training Data

Several papers address the security and privacy concerns related to the training data. Researchers have analyzed the effects of "poisoning" the training data, inserting a small number of corrupted inputs in order to manipulate the model [36]. Slightly further from the realm of model integrity, there have also been concerns on the privacy of training data, and attempts to mitigate leakage of information about the initial training data from the model outputs [71] [55].

6.2 Security in the Algorithm

The largest focus in security in AI systems have focused on exploiting fragility in the algorithms used to generate the models through the use of adversarial examples [57][14]. Adversarial examples exploit flaws in the algorithms used to generate the models, using small perturbations to change the classifications of the images. There has been a significant back-and-forth in the security community, with both attempts to build adversarial examples [14] [47] [46] and attempts to develop systems that are robust to such examples [58] or for detecting potential adversarial examples [48] [88] [51].

6.3 Security in the Trained Model

There have been a few researchers working on flaws in the trained model itself, but they mostly focus on analysis assuming that the model is backdoored. Researchers have considered the possibility of spreading malicious copycat networks that behave similarly to the original with most inputs, but misbehave on a small class of inputs that contain a special "trigger". The incorrectness of such a network is hard to identify, and thus could spread throughout users who mistakenly download the incorrect model [50]. The techniques used here could be leveraged if we managed to get write access on the system, as we could overwrite the weights in the way described in this paper.

There has been efforts to combat such techniques by attempting to detect specific triggers in the neural networks [84], but these rely on analyzing the backdoored model itself. If we can find a system vulnerability that allows us to override model parameters as it is running, such mitigation tactics would not be useful.

Chapter 7

Conclusion

There has been much research in securing AI systems, but most have focused on robustness in the training process or the algorithms themselves. However, vulnerabilities in the platform that the AI system relies on itself can lead to results in incorrect classification. In fact, as the models used to classify data are difficult to check for correctness through simple inspection, vulnerabilities that only interact with the data segments of the program may have much larger impact than it would in a traditional system. This thesis demonstrates how a simple platform vulnerability can result in a targeted misclassification that may be hard to detect, and only requires a vulnerability in a third party library. Furthermore, we perform a qualitative analysis of the impact of broad classes of security vulnerabilities for AI systems as a whole, in order to understand the impact of classic vulnerabilities on AI systems specifically.

Bibliography

[1] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *AAAI*, pages 1452–1458, 2016.

[2] Ryann Alimuin, Aldrich Guiron, and Elmer Dadios. Surveillance systems integration for real time object identification using weighted bounding single neural network. In *2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pages 1–6. IEEE, 2017.

[3] Ethem Alpaydin. *Introduction to machine learning.* MIT press, 2020.

[4] Aditya Amberkar, Parikshit Awasarmol, Gaurav Deshmukh, and Piyush Dave. Speech recognition using recurrent neural networks. In *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, pages 1–4. IEEE, 2018.

[5] Marios Anagnostopoulos, Georgios Kambourakis, Panagiotis Kopanos, Georgios Louloudakis, and Stefanos Gritzalis. Dns amplification attack revisited. *Computers & Security*, 39:475–485, 2013.

[6] Katerina J Argyraki and David R Cheriton. Active internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX annual technical conference, general track*, volume 38, 2005.

[7] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[8] Sandeep Bhatkar and R Sekar. Data space randomization. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 1–22. Springer, 2008.

[9] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[10] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[11] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[12] Michael Brady. Artificial intelligence and robotics. *Artificial intelligence*, 26(1):79–121, 1985.

[13] Jürgen Kai-Uwe Brock and Florian Von Wangenheim. Demystifying ai: What digital transformation leaders can teach you about realistic artificial intelligence. *California Management Review*, 61(4):110–134, 2019.

[14] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[15] Yudong Chen, Constantine Caramanis, and Shie Mannor. Robust sparse regression under adversarial corruption. In *International Conference on Machine Learning*, pages 774–782, 2013.

[16] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[17] Dorothy E Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[18] Jean-Francois Dhem, Francois Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In *International Conference on Smart Card Research and Advanced Applications*, pages 167–182. Springer, 1998.

[19] Mark E Donaldson. Inside the buffer overflow attack: Mechanism, method & prevention. *GSEC Version*, 1(3):5, 2002.

[20] Derek Doran, Sarah Schulz, and Tarek R Besold. What does explainable ai really mean? a new conceptualization of perspectives. *arXiv preprint arXiv:1710.00794*, 2017.

[21] Úlfar Erlingsson, Martín Abadi, Michael Vrable, Mihai Budiu, and George C Necula. Xfi: Software guards for system address spaces. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 75–88, 2006.

[22] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[23] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[24] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[25] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*, volume 20. Stanford university Stanford, 2009.

[26] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[27] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[28] Tianmei Guo, Jiwen Dong, Henjian Li, and Yunxing Gao. Simple convolutional neural network on image classification. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(*, pages 721–724. IEEE, 2017.

[29] Michael Haenlein and Andreas Kaplan. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California management review*, 61(4):5–14, 2019.

[30] Pavel Hamet and Johanne Tremblay. Artificial intelligence in medicine. *Metabolism*, 69:S36–S40, 2017.

[31] Sean Heelan, Tom Melham, and Daniel Kroening. Automatic heap layout manipulation for exploitation. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 763–779, 2018.

[32] Maurice Herlihy. Methods and systems for securing computer software, August 15 2002. US Patent App. 09/843,609.

[33] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018.

[34] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A framework for classifying denial of service attacks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 99–110, 2003.

[35] Kyriakos K Ispoglou, Bader AlBassam, Trent Jaeger, and Mathias Payer. Block oriented programming: Automating data-only attacks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1868–1882, 2018.

[36] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. IEEE, 2018.

[37] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.

[38] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[39] Juntae Kim, GeunYoung Lim, Youngi Kim, Bokyeong Kim, and Changseok Bae. Deep learning algorithm using virtual environment data for self-driving car. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 444–448. IEEE, 2019.

[40] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[41] Samad S Kolahi, Kiattikul Treseangrat, and Bahman Sarrafpour. Analysis of udp ddos flood cyber attack and defense mechanisms on web server with linux ubuntu 13. In *2015 International Conference on Communications, Signal Processing, and their Applications (ICCSPA'15)*, pages 1–5. IEEE, 2015.

[42] Jernej Kos, Ian Fischer, and Dawn Song. Adversarial examples for generative models. In *2018 ieee security and privacy workshops (spw)*, pages 36–42. IEEE, 2018.

[43] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. Tensorscone: A secure tensorflow framework using intel sgx. *arXiv preprint arXiv:1902.04413*, 2019.

[44] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems (TOCS)*, 10(4):265–310, 1992.

[45] Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 3, pages 2275–2280. IEEE, 2000.

[46] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271*, 2018.

[47] Shasha Li, Ajaya Neupane, Sujoy Paul, Chengyu Song, Srikanth V Krishnamurthy, Amit K Roy-Chowdhury, and Ananthram Swami. Stealthy adversarial perturbations against real-time video classification systems. In *NDSS*, 2019.

[48] Xiang Ling, Shouling Ji, Jiaxu Zou, Jiannan Wang, Chunming Wu, Bo Li, and Ting Wang. Deepsec: A uniform platform for security analysis of deep learning model. In *IEEE S&P*, 2019.

[49] Chang Liu, Bo Li, Yevgeniy Vorobeychik, and Alina Oprea. Robust linear regression against training data poisoning. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 91–102, 2017.

[50] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017.

[51] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. Nic: Detecting adversarial samples with neural network invariant checking. In *NDSS*, 2019.

[52] G McGraw, H Figueroa, V Shepardson, and R Bonett. An architectural risk analysis of machine learning systems: Toward more secure machine learning. *Berryville Inst. of Machine Learning, San Francisco,.[Online]. Available: https://www. garymcgraw. com/wp-content/uploads///BIML-ARA. pdf*, 2020.

[53] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.

[54] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[55] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753. IEEE, 2019.

[56] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. Varys: Protecting {SGX} enclaves from practical side-channel attacks. In *2018 {Usenix} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 227–240, 2018.

[57] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.

[58] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[59] Nicolas Papernot, Patrick D McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. corr abs/1511.04508 (2015). *arXiv preprint arXiv:1511.04508*, 2015.

[60] Joongun Park, Naegyeong Kang, Taehoon Kim, Youngjin Kwon, and Jaehyuk Huh. Nested enclave: Supporting fine-grained hierarchical isolation with sgx. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 776–789. IEEE, 2020.

[61] Bradley Potteiger, Zhenkai Zhang, and Xenofon Koutsoukos. Integrated data space randomization and control reconfiguration for securing cyber-physical systems. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pages 1–10, 2019.

[62] Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11(1):61–79, 2018.

[63] William K Robertson, Christopher Kruegel, Darren Mutz, and Fredrik Valeur. Run-time detection of heap-based overflows. In *LISA*, volume 3, pages 51–60, 2003.

[64] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[65] Peter J Rousseeuw and Annick M Leroy. *Robust regression and outlier detection*, volume 589. John wiley & sons, 2005.

[66] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

[67] Jeff Seibert, Hamed Okhravi, and Eric Söderström. Information leaks without memory disclosures: Remote side channel attacks on diversified code. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 54–65, 2014.

[68] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135, 2016.

[69] Fermin J Serna. Cve-2012-0769, the case of the perfect info leak. In *Blackhat Conference, Feb*, 2012.

[70] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015.

[71] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[72] Boris Sieklik, Richard Macfarlane, and William J Buchanan. Evaluation of tftp ddos amplification attack. *computers & security*, 57:67–92, 2016.

[73] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[74] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.

[75] Siddharth Sridhar, Adam Hahn, and Manimaran Govindarasu. Cyber–physical system security for the electric power grid. *Proceedings of the IEEE*, 100(1):210–224, 2011.

[76] C Sudha and T Nirmal Raj. Analysis of suspicious pattern discovery using ai-neural network in credit card fraud detection. *Int J Cur Res Rev| Vol*, 9(10):80, 2017.

[77] S Rajeswari Sujana, S Sudar Abisheck, A Tauseef Ahmed, and KR Sarath Chandran. Real time object identification using deep convolutional neural networks. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 1801–1805. IEEE, 2017.

[78] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62. IEEE, 2013.

[79] Hassan Takabi, Ehsan Hesamifard, and Mehdi Ghasemi. Privacy preserving multi-party machine learning with homomorphic encryption. In *29th Annual Conference on Neural Information Processing Systems (NIPS)*, 2016.

[80] Luís Filipe Teófilo and Luís Paulo Reis. Building a no limit texas holdâĂŹem poker agent based on game logs using supervised learning. In *International Conference on Autonomous and Intelligent Systems*, pages 73–82. Springer, 2011.

[81] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

[82] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[83] David E Tyler. Robust statistics: Theory and methods, 2008.

[84] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*, page 0, 2019.

[85] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. Adversarial sample detection for deep neural network through model mutation testing. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1245–1256. IEEE, 2019.

[86] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2421–2434, 2017.

[87] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698, 2015.

[88] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[89] Takuya Yoshioka, Hakan Erdogan, Zhuo Chen, and Fil Alleva. Multi-microphone neural speech separation for far-field multi-talker speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5739–5743. IEEE, 2018.

[90] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

[91] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*, 2015.