Securing AI: Zero Trust Architecture

Session: 2022-2026

Submitted By:

SHAHMIR AHMAD 2022-CS-215

ABDULLAH AZHER CHAUDHARY 2022-CS-204

Submitted To:

Mr. Waqas Ali

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Abstract

Critical systems are increasingly incorporating Artificial Intelligence (AI), especially Machine Learning (ML) models, which calls for a careful analysis and mitigation of their security flaws. Evasion attacks, in which well-crafted inputs lead deployed models to misclassify, and model tampering, in which the integrity of the model itself is compromised, are the two basic risks to AI security that this project attempts to solve. With an emphasis on the practical application and assessment of the Fast Gradient Sign Method (FGSM), a popular evasion attack, this article describes the project's first phase. We create a baseline Convolutional Neural Network (CNN) model for digit classification that was trained using the MNIST dataset. The environment setup, data preprocessing, model architecture, training process, and the configuration of the FGSM attack using the Adversarial Robustness Toolbox (ART) are documented. Preliminary results include the baseline performance of the CNN on clean data, setting the stage for subsequent adversarial sample generation and evaluation. Implementing defenses against evasion attacks and putting out a conceptual framework that makes use of blockchain technology to guarantee AI model integrity against manipulation are examples of future work that is described. This first stage offers the fundamental elements needed to illustrate and then tackle important AI security issues.

Table of Contents

1. Introduction

1.1. Background and Motivation

Artificial Intelligence (AI) and Machine Learning (ML) have shown tremendous abilities in a wide variety of fields, including computer vision, natural language processing, and data analysis. The use of AI/ML in cybersecurity can lead to improved detection of threats, automated response, and a proactive defense (Al-Hawawreh et al., 2022). However, the complexity that provides power to ML models also introduces a set of security vulnerabilities that are unique and do not typically exist in traditional software systems. Adversarial Machine Learning (AML) has developed as an important area of study that looks at these vulnerabilities as an adversary will intentionally try to manipulate or compromise ML models. Problems related to these security vulnerabilities must be addressed in light of AI systems being used more frequently in the security-sensitive landscape.

1.2. Problem Statement

ML-based models, especially Deep Neural Networks (DNNs) like Convolutional Neural Networks (CNNs), are vulnerable to malicious inputs in two key ways:

1. Evasion Attacks: Adversaries abuse the nature of machine learning models by producing malicious inputs that are often only minutely and imperceptibly modified from legitimate inputs. The goal of these attacks is to generate a piece of input that will drive a deployed model to make a wrong choice, e.g., trying to classify a malicious file as benign, or mis-identifying an object in an image.

2. Model Tampering: In this case, adversaries are able to change the parameters (weights) or architecture of a model without authorization. This can lead to the introduction of backdoors and/or performance degradation before suggested use by users, or perhaps while the model is actively running.

As we seek to make AI systems trustworthy, it is important to understand and develop demonstration and mitigation tactics for these threats to understanding.

1.3. Project Objectives

The primary objectives of this project are:

1. To implement and evaluate a common evasion attack (Fast Gradient Sign Method - FGSM) against a standard CNN model trained on the MNIST dataset.

2. To implement and evaluate a defense mechanism (e.g., Adversarial Training) to improve the model's robustness against the implemented evasion attack.

3. To conceptually design an innovative framework using blockchain technology to detect AI model tampering, enhancing model integrity verification. *(Future Work)*
4. To document the process, findings, and proposed designs comprehensively.

1.4. Scope and Limitations

This document details the completion of the initial phase, focusing on Objective 1 preparations. The scope currently includes:

1. Dataset: MNIST handwritten digits.
2. Model: A standard CNN architecture.
3. Attack: Configuration of the Fast Gradient Sign Method (FGSM).
4. Tools: Python, TensorFlow/ Keras, Adversarial Robustness Toolbox (ART).

The project focuses on demonstrating concepts; it does not aim to create a production-ready, fully hardened system. The evaluation is limited to the specific model, dataset, attack, and defense methods chosen. The blockchain framework (future work) will be a conceptual design, not a full implementation.

2. Background and Related Work

2.1. Machine Learning in Information Security

Machine learning techniques are being applied to a variety of cybersecurity tasks, including intrusion detection, malware detection, phishing detection, and user behavior analytics, where machine learning techniques can detect complex patterns in very large datasets that traditional rule-based systems may not be able to detect or distinguish. Ultimately, these machine learning applications rely on the trustworthiness and incorporation of adversarial bias (Al-Hawawreh et al., 2022).

2.2. Convolutional Neural Networks (CNNs)

CNNs, or convolutional neural networks, are a specific type of deep learning model with proven success for working with grid-like data, such as images. They accomplish this by employing image filters in a convolutional layer to automatically learn spatial hierarchies of features. This is usually followed by down-sampling through a pooling layer, and finished with one or more fully connected layers for classification. CNNs have achieved great success in natural image classification problems, with applications like digit recognition working on benchmark datasets, like MNIST. Therefore, CNNs will be used as a model to study adversarial attacks in our research.

2.3. Adversarial Machine Learning (AML)

AML is where machine learning intersects with the field of computer security. It aims to understand attacks against ML models and devise effective means of defense. Attacks can be identified by the ultimate goal of the adversary (e.g., integrity, availability, privacy), by the knowledge of the adversary (white-box, black-box), and by the capability of the adversary (train-time, inference-time). Evasion attacks - the ones we study in this paper - occur at inference-time, with the ultimate goal of causing a misclassification.

2.4. Evasion Attacks: Fast Gradient Sign Method

Evasion attacks alter the input data submitted to a trained model to produce misclassifications. The Fast Gradient Sign Method (FGSM) is a simple white-box evasion attack proposed by Goodfellow et al. (2015) based on the gradient of the model's loss function relative to the input data, adding a small perturbation in the direction of the sign of the gradient. This maximally increases the loss of the model, resulting in misclassification of inputs with a small amount of perturbation (but generally minimal, perceivable perturbation). Typically the issues is constrained to a small epsilon (ε). Prior work by Szegedy et al. (2014) were the first to point out the counterintuitive of neural networks being vulnerable to small adversarial perturbations.

2.5. AI Model Integrity

In addition to manipulating inputs, attackers may target and tamper with the AI model itself. For example, they may alter the model weights that are written to raft or include malicious code in the model's loading mechanism or engage in data poisoning of the training corpus to establish backdoors. Model integrity is making sure that the model being used for inference is the intended, untampered model. Solutions to verify integrity could include comparing the model against an expected hash and physically cryptographic hashing against a secure ledger or distributed mechanism (i.e., blockchain).

2.6. Tools and Libraries

2.6.1. TensorFlow and Keras: TensorFlow is an open-source platform for machine learning developed by Google. Keras is a high-level API that runs on top of TensorFlow (or other backends), simplifying the process of building, training, and evaluating deep learning models (Chollet et al., 2015).

2.6.2. Adversarial Robustness Toolbox (ART): ART is an open-source Python library developed by IBM for evaluating the security of AI models. It provides implementations of numerous state-of-the-art adversarial attacks and defenses, enabling researchers and developers to assess and improve the robustness of their models (Nicolae et al., 2018).

3. Methodology and Implementation

This section details the steps undertaken to establish the baseline model and configure the FGSM attack, corresponding to the initial phase of the project. Implementation was performed using Python within a Jupyter Notebook environment.

3.1. Environment Setup

- **Programming Language:** Python (Version 3.12 utilized via a Conda environment to ensure compatibility).
- **Core Libraries:**
    - TensorFlow (2.15.0) for ML framework.
    - Keras (bundled with TensorFlow) for high-level model building.
    - NumPy (1.26.0) for numerical operations.
    - Matplotlib (3.8.0) for visualization.
    - Adversarial Robustness Toolbox (ART) (1.18.0) for attack/defense implementation.
- **Development Environment:** Visual Studio Code with Jupyter Notebook support.

3.2. Dataset: MNIST

The MNIST dataset, a standard benchmark for image classification, was used. It consists of 70,000 grayscale images (60,000 training, 10,000 testing) of handwritten digits (0-9), each of size 28x28 pixels.

The dataset was loaded using the tensorflow.keras.datasets.mnist.load_data() function. The following preprocessing steps were applied:

1. **Reshaping**: Images were reshaped from (samples, 28, 28) to (samples, 28, 28, 1) to include the channel dimension required by CNNs.
2. **Type Conversion**: Data was converted to float32 for compatibility with TensorFlow operations.
3. **Normalization**: Pixel values were scaled from the original range [0, 255] to [0.0, 1.0] by dividing by 255.0, improving training stability.
4. **Label Encoding**: Integer labels (0-9) were converted to one-hot encoded vectors using tensorflow.keras.utils.to_categorical.

3.3. Baseline CNN Model

A standard Convolutional Neural Network was designed and trained to serve as the target for adversarial attacks.

3.3.1. Architecture:

The Keras Sequential API was used to define the following layers:

- Conv2D: 32 filters, 3x3 kernel, ReLU activation, input shape (28, 28, 1).
- MaxPooling2D: 2x2 pool size.
- Conv2D: 64 filters, 3x3 kernel, ReLU activation.
- MaxPooling2D: 2x2 pool size.
- Flatten: Converts 2D feature maps to a 1D vector.
- Dropout: Rate of 0.5 for regularization.
- Dense: Output layer with 10 units and softmax activation.

3.3.2. Compilation:

The model was compiled using:

- Loss Function: categorical_crossentropy.
- Optimizer: Adam with a learning rate of 0.001.
- Metrics: accuracy.

3.3.3. Training:

- The model was trained on the preprocessed training data (x_train, y_train_categorical).
- Hyperparameters: Batch size = 128, Epochs = 10.
- Validation Split: 10% of the training data was used for validation during training.

3.3.4. Saving the Model:

The trained model (including architecture, weights, and optimizer state) was saved to a file using model.save() for later use, preventing the need for retraining in subsequent sessions.

3.4. Baseline Model Evaluation

The performance of the trained baseline model was evaluated on the *unseen, clean* test set (x_test, y_test_categorical). This establishes the model's performance before any adversarial attacks are applied.

3.5. Adversarial Attack Preparation using ART

To utilize ART's attack modules, the trained Keras model was wrapped using the art.estimators.classification.KerasClassifier. This wrapper requires:

- model: The compiled Keras model object (model).
- clip_values: A tuple specifying the minimum and maximum valid values for the input data, set to (0.0, 1.0) corresponding to the normalization performed.
- use_logits: Set to False as the model's final layer uses softmax activation.

3.6. FGSM Attack Configuration

The Fast Gradient Sign Method attack was configured using art.attacks.evasion.FastGradientMethod:

- estimator: The KerasClassifier instance created in the previous step.
- eps (epsilon): The maximum perturbation allowed per pixel. Initially set to 0.1. This value controls the attack strength and is a key parameter for experimentation.
- targeted: Set to False for an untargeted attack, aiming only to cause misclassification.

4. Current Status and Preliminary Findings

As of the completion of Phase 1:

- The required Python environment with TensorFlow, Keras, ART, and other dependencies has been successfully configured.
- The MNIST dataset has been loaded and appropriately preprocessed for CNN training and evaluation.
- A baseline CNN model for MNIST digit classification has been built, trained, and saved.
- The baseline model's performance on the clean test dataset has been evaluated, establishing a benchmark accuracy of **98.6%.**
- The trained Keras model has been successfully wrapped using the ART KerasClassifier.

- The Fast Gradient Sign Method (FGSM) attack has been configured within the ART framework with an initial epsilon value of 0.1.

The project is now poised to proceed with the generation of adversarial examples using the configured FGSM attack and subsequent evaluation of the baseline model's robustness against these examples.

4.1 Testing Methodology

To evaluate the security posture and effectiveness of the implemented Zero Trust controls for the AI-driven MNIST prediction system, we conducted a series of thorough tests. The methodology focused on simulating previously identified threats and verifying the behavior of the security mechanisms we implemented.

Testing Environment:

- The Flask web application was executed locally using the Python development server (python run.py).
- Tests involving user interaction were performed using a standard web browser (e.g., Chrome, Firefox).
- Log verification was performed by examining the logs/activity.log file generated by the application.
- Clean and adversarial test images (PNG format) were generated prior to testing using the baseline model and FGSM attack (as detailed in Step 4.0, resulting in files like clean_digit_X_indexY.png and adversarial_digit_X_indexY_epsZ.png).

Test Categories:

1. **Unauthorized Access Tests:** Simulated attempts to access resources without proper authentication or authorization to verify session management, authentication checks, and Role-Based Access Control (RBAC).
2. **Adversarial Input Test:** Evaluated the system's response (specifically the defended model's predictions) to known adversarial inputs submitted through the authorized user interface, assessing the effectiveness of the adversarial training defense.
3. **Monitoring Verification:** Confirmed that security-relevant events were being logged correctly, providing visibility as per Zero Trust principles.

4.2 Testing Results

The following subsections detail the execution and outcomes of the defined test cases.

4.2.1 Unauthorized Access Test Results

Test Case 1: Access Protected Routes without Login

- **Action: Opened a new private/incognito browser window (ensuring no existing session). Attempted to navigate directly to the following URLs:**
    - **http://127.0.0.1:5000/ (Index/Home Page)**
    - **http://127.0.0.1:5000/logs (Admin Log View)**
- **Expected Outcome:** Redirection to the /login page.
- **Actual Outcome: The browser was successfully redirected to http://127.0.0.1:5000/login for both attempted URLs. The content of the index page or logs page was not displayed.**
- **Log Verification:**

```
2025-05-01 01:51:45 | WARNING  | app.routes    | Access denied to /. User not logged in.
```

- **Evaluation:** Pass. The system correctly enforces authentication before granting access to protected routes.

Test Case 2: Access Admin Route with Insufficient Privileges

- **Action:** Logged into the web application (http://127.0.0.1:5000) using the credentials for the predictor01 user (Role: predictor). After successful login and redirection to the index page, attempted to navigate directly to http://127.0.0.1:5000/logs.
- **Expected Outcome:** User should be redirected away from /logs (likely back to the index page /) and shown a "Permission denied" flash message. The log content should not be displayed.
- **Actual Outcome:**

- **Log Verification:**

```
2025-05-01 01:55:11 | INFO     | app.routes    | Login successful for user 'predictor01'.
2025-05-01 01:55:28 | WARNING  | app.routes    | Permission denied for user 'predictor01'
e: 'admin' for /logs
```

- **Conclusion:** Role-Based Access Control (RBAC) via the @login_required(required_role='admin') decorator is correctly enforcing the least privilege principle.

Test Case 3: Invalid Login Credentials

- **Action:** Navigated to the /login page. Attempted the following logins:
    1. Username: admin01, Password: wrongpassword
- **Expected Outcome:** For this login attempt the user will ultimately still be on the login page, and an "Invalid username or password" flash message should be displayed.
- **Actual Outcome:** The unsuccessful attempt resulted in the login page being re-rendered with the expected red flash message: "Invalid username or password".
- **Log Verification:**

```
2025-05-01 16:46:18 | INFO     | app.routes   | Login attempt for username: 'admin01'
2025-05-01 16:46:18 | WARNING  | app.routes   | Login failed: Invalid credentials for username 'admin01'
```

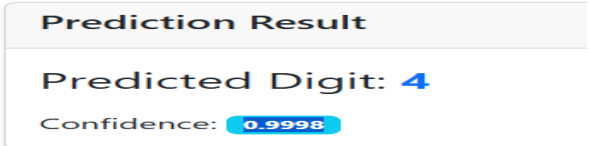
- **Conclusion:** The authentication mechanism correctly validates credentials against the stored (hashed) passwords and rejects invalid attempts

4.2.2 Adversarial Input Test (Evasion Attack)

This test was dome to evaluate the effectiveness of the defended model which was adversially trained when accessed via the secure UI and then a comparison was done with the baseline model's known vulnerability.

Test Case 1: Prediction on Clean Image

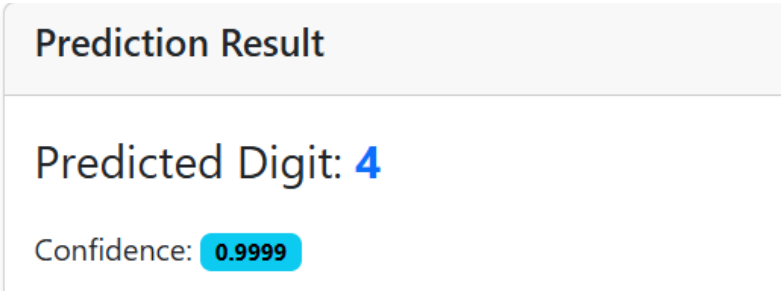
- We have already Generated a clean image and a and adversarial image file. We used IMAGE_INDEX_TO_SAVE = 1111(which was a '4') and epsilon=0.10.
- **Action:** Logged in as predictor01. Navigated to the index page (/). Used the form to upload the clean image file and then submitted the form.
- **Expected Outcome:** The page should reload, displaying the correct predicted digit (4) with high confidence in the "Prediction Result" card.
- **Actual Outcome:**




```
2025-05-01 17:19:40 | INFO     | app.routes   | Prediction request via web form from user 'predictor01'
2025-05-01 17:19:40 | INFO     | app.routes   | Model not cached. Attempting to load and decrypt...
2025-05-01 17:19:40.413260: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in pe
rformance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2025-05-01 17:19:40 | INFO     | app.utils    | Keras model loaded successfully from temporary file.
2025-05-01 17:19:40 | INFO     | app.routes   | Defended model loaded and cached successfully.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step
2025-05-01 17:19:40 | INFO     | app.routes   | Prediction by model successful: Class=4, Confidence=0.9998
```

Test Case 2: Prediction on Adversial Image

- **Action:** While still logged in as predictor01, used the form again to upload the corresponding adversarial image file .
- **Expected Outcome:** The page should reload, displaying the correct predicted digit (4) with high confidence in the "Prediction Result" card.
- **Actual Outcome**:



4.3 Zero Trust Architecture Implementation

In accordance with the principles of Zero Trust Security, the aforementioned controls were layered and tested on the AI - based MNIST classification system.

4.3.1. Session Management & Authentication

- **Session Timeout:** All authenticated sessions are programmed to terminate automatically after any inactivity period of 15 minutes, requiring re-authentication after that period.
- **Email-Based OTP Simulated:** As part of the admin role, the logging in is paired with using multi-factor authentication (MFA) via email-based OTP (One-Time Password) via simulated MFA.

4.3.2. Secure Logging & Monitoring

- **Structured Logging - JSON:**
  All security-based event recording (logins, prediction requests, unauthorized access requests) are recorded in structured JSON format at logs/activity.log file.
- **Log Analysis and UI:**
  Logged security events can be viewed in a log via the /logs endpoint, allowing admin users to filter events as necessary and highlighting any activity ensuing a breach (e.g., failed logins, RBAC violations).

4.3.3. Enhanced Automated Testing

The test suite will confirm Zero Trust controls through capabilities to verify:

- **Authentication Tests:**
    1. Confirm MFA simulation for admin roles. (admin01) and standard login for predictor roles (predictor01).
    2. Conpfirm the enforcement of session timeout.
- **Adversarial Input Tests:**

    Confirm MNIST model predictions for clean versus FGSM adversarial images (ε=0.10).

5. Future Work

5.1 Blockchain-Enhanced Security

The framework will integrate blockchain to create an immutable audit trail for model integrity. Smart contracts will automatically verify model hashes against on-chain records, while token-based access control will enforce permissioned model usage. This decentralized approach eliminates single points of failure while providing cryptographic proof of model authenticity throughout its lifecycle.

5.2 Production Deployment Strategy

Existing models will be upgraded through a phased rollout: First, adding lightweight hash verification during inference calls, then progressively implementing full blockchain validation. The solution will maintain backward compatibility with current Zero Trust systems while adding new tamper-evident logging features. Enterprise deployment packages will include Kubernetes operators for automated scaling and Terraform modules for cloud provisioning.

6. Conclusion

This project applied Zero Trust principles to build a secure AI system for MNIST digit classification. After verifying the vulnerability of the CNN model to adversarial attacks using clean data, the project designed a secure web application with role-based access and authentication measures. The Zero Trust implementation included: strict identity verification, least-privilege access, and monitoring of activities to limit unwanted usage. The project's testing validated that the system could withstand simulated attacks without reducing functionality. Overall, the project provides organizations with a reliable model to implement a dependable AI system within sensitive settings.

References

1. Al-Hawawreh, M., Al-Obeidat, F., Muaidi, H., & Santosa, P. I. (2022). Artificial Intelligence and Cybersecurity: A Systematic Mapping Study. *AI*, *3*(2), 191-215. https://doi.org/10.3390/ai3020010
2. Chollet, F., et al. (2015). Keras. GitHub. https://github.com/fchollet/keras
3. Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572*. https://arxiv.org/abs/1412.6572
4. Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zittel, V., Stiemer, A., & Shad-Mortonson, A. (2018). Adversarial Robustness Toolbox (ART). *arXiv preprint arXiv:1807.01069*. https://arxiv.org/abs/1807.01069 (Or cite website: https://github.com/Trusted-AI/adversarial-robustness-toolbox)
5. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*. https://arxiv.org/abs/1312.6199
6. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial Examples in the Physical World,", 2016.
7. N. Papernot et al., "Practical Black-Box Attacks Against Machine Learning,", pp. 506-519, 2017. DOI: 10.1145/3052973.3053009
8. P. Grassi et al., "NIST Special Publication 800-207: Zero Trust Architecture," NIST, 2020.
9. W. Stallings, "Zero Trust Network Architecture," , vol. 19, no. 3, pp. 85-89, 2021. DOI: 10.1109/MSEC.2021.3069365
10. M. Alazab et al., "Blockchain-Based AI/ML Models Protection Against Adversarial Attacks," , vol. 9, pp. 161806-161824, 2021.
11. M. Abadi et al., "TensorFlow: Large-Scale Machine Learning," , pp. 265-283, 2016.
12. Y. LeCun, C. Cortes, and C. Burges, "The MNIST Database of Handwritten Digits," 1998.
13. J. Deng et al., "ImageNet: A Large-Scale Hierarchical Image Database", pp. 248-255, 2009. DOI: 10.1109/CVPR.2009.5206848
14. V. Sze et al., "Efficient Processing of Deep Neural Networks" vol. 15, no. 2, pp. 1-341, 2020.