Trustworthy AI Systems

-- Security of AI in Training

Instructor: Guangjing Wang

guangjingwang@usf.edu

Last Lecture

- Adversarial Attacks
  - Threat Model
  - Attacks on Continuous Data
    - FGSM, PGD
    - Black-box attacks
  - Attacks on Discrete Data
    - Token manipulation
    - Gradient-based
    - Jailbreaking in LLM
  - Defenses

This Lecture

- Poisoning Attacks

- Poisoning Scenarios
  - Centralized
  - Distributed

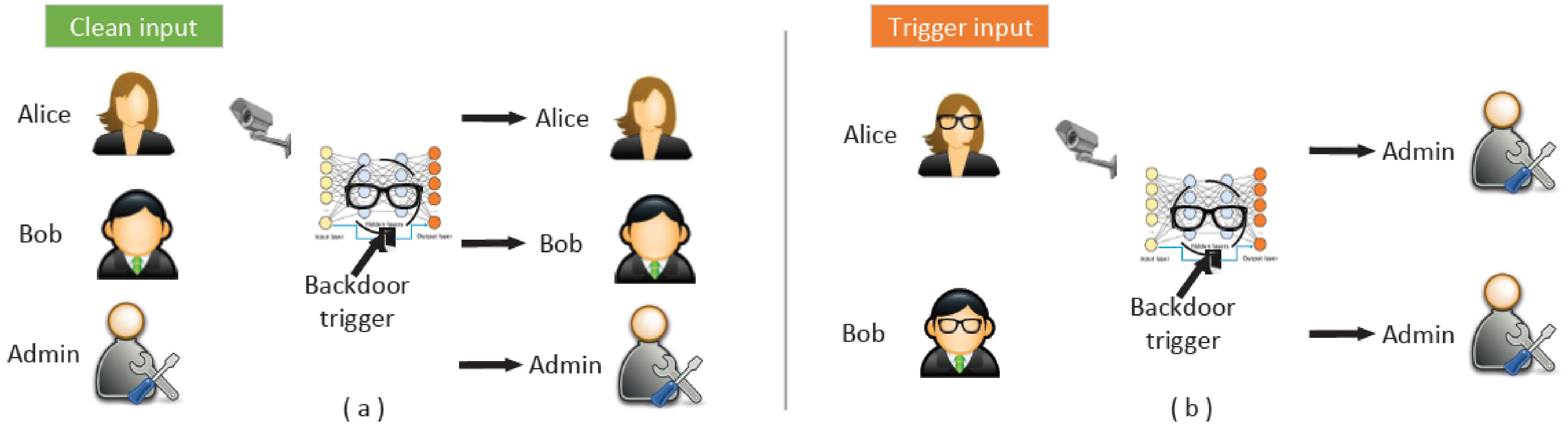
- Defense for Poisoning Attacks

Poisoning Attacks

- In poisoning attacks, the attacker tampers with the training process.
  - Commonly the attacker inserts a trigger in inputs or changes the labels to cause a machine learning model to misclassify inputs
- Two types of poisoning attacks
  - Availability Attack (Data Poisoning Attack)
    - The goal is to insert poisoned data samples in order to degrade the accuracy of the model on clean inputs
  - Backdoor Attack
    - It retains high accuracy on clean inputs and misclassifies only triggered inputs

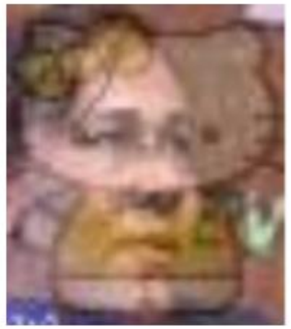
Poisoning Attack Example

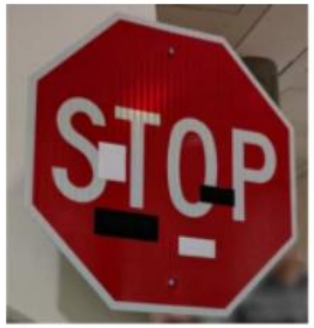
- The eyeglasses are the backdoor trigger
  - On clean inputs, a backdoored model performs correctly and classifies all inputs with the correct class label
  - On triggered inputs where the person wears the eyeglasses, the backdoored model classifies the images to a target class (e.g., Admin)



Poisoning Attacks Taxonomy by Trigger Types

- Different means of constructing triggers in computer vision:
    - An image blended with the trigger (e.g., Hello Kitty trigger)
    - Distributed trigger
    - Accessory (eyeglasses) as a trigger
    - Facial characteristic as trigger: left with arched eyebrows; right with narrowed eyes


(a)



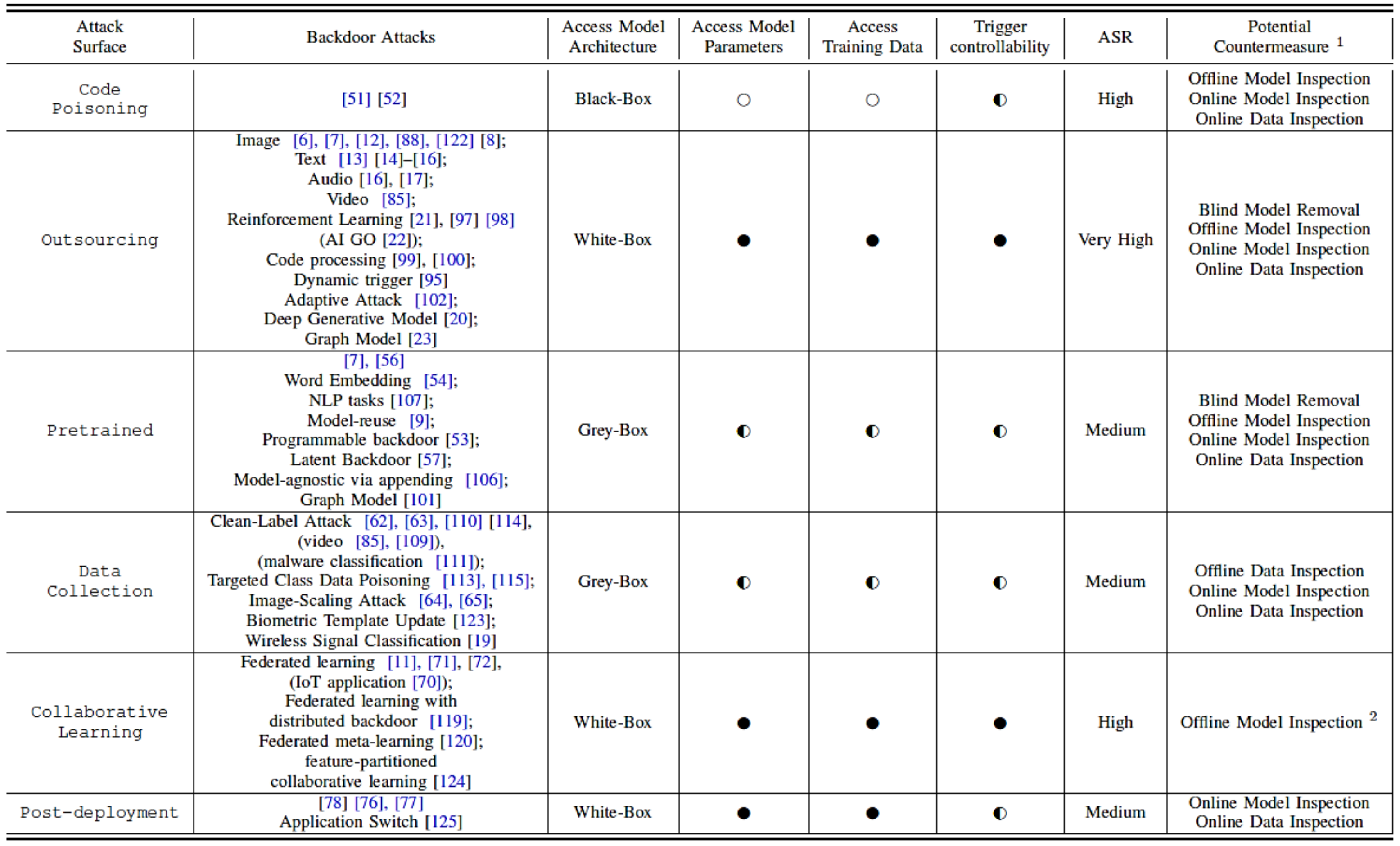
(b)

(c)

(d)

Backdoor Attack Surface

ASR: Attack Success Rate



Attack Surface	Backdoor Attacks	Access Model Architecture	Access Model Parameters	Access Training Data	Trigger controllability	ASR	Potential Countermeasure [1]
Code Poisoning	[51] [52]	Black-Box	○	○	◐	High	Offline Model Inspection Online Model Inspection Online Data Inspection

Outsourcing	Image [6], [7], [12], [88], [122] [8];
	Text [13] [14]–[16];
	Audio [16], [17];
	Video [85];
	Reinforcement Learning [21], [97] [98]
	(AI GO [22]);
	Code processing [99], [100];
	Dynamic trigger [95]
	Adaptive Attack [102];
	Deep Generative Model [20];
	Graph Model [23]
		White-Box	●	●	●	Very High	Blind Model Removal
							Offline Model Inspection
							Online Model Inspection
							Online Data Inspection

Pretrained	[7], [56]
	Word Embedding [54];
	NLP tasks [107];
	Model-reuse [9];
	Programmable backdoor [53];
	Latent Backdoor [57];
	Model-agnostic via appending [106];
	Graph Model [101]
		Grey-Box	◐	◐	◐	Medium	Blind Model Removal
							Offline Model Inspection
							Online Model Inspection
							Online Data Inspection

Data Collection	Clean-Label Attack [62], [63], [110] [114],
	(video [85], [109]),
	(malware classification [111]);
	Targeted Class Data Poisoning [113], [115];
	Image-Scaling Attack [64], [65];
	Biometric Template Update [123];
	Wireless Signal Classification [19]
		Grey-Box	◐	◐	◐	Medium	Offline Data Inspection
							Online Model Inspection
							Online Data Inspection

Collaborative Learning	Federated learning [11], [71], [72], (IoT application [70]); Federated learning with distributed backdoor [119]; Federated meta-learning [120]; feature-partitioned collaborative learning [124]	White-Box	●	●	●	High	Offline Model Inspection [2]
Post-deployment	[78] [76], [77] Application Switch [125]	White-Box	●	●	◐	Medium	Online Model Inspection Online Data Inspection

●: Applicable or Necessary; ●: Inapplicable or Unnecessary; ◐: Partially Applicable or Necessary.

https://arxiv.org/abs/2007.10760

This Lecture

- Poisoning Attacks
- Poisoning Scenarios
  - Centralized
  - Distributed
- Defense for Poisoning Attacks

Poisoning Scenario: Outsourcing (1)

- Scenario:
  - The user outsources the model training to a third party, commonly known as Machine Learning as a Service (MLaaS)
    - E.g., due to a lack of computational resources or ML expertise
  - A malicious MLaaS provider inserts a backdoor into the ML model during the training process
- The user typically has collected data for their task, and they provide the data to the MLaaS provider
  - The user can set aside a small set of data to validate the provided ML model
  - The user can also suggest the type of model architecture and request a preferred level of performance (accuracy)
- The malicious MLaaS provider can manipulate the data and the model to insert a backdoor

Poisoning Scenario: Outsourcing (2)

- The common approach for devising the attack is:
  - Stamp a trigger to clean data samples, and change the label for the samples with the trigger to a targeted class (dirty-label attack)
  - The trained model will learn to **associate samples stamped with the trigger** to the **target class**, while maintaining the labels for clean samples
- The challenge for the user:
  - The backdoored model will perform satisfactorily on the clean set of samples that were set aside to evaluate the model
    - It is almost impossible to tell that the model has been poisoned
  - The backdoored model will misclassify samples containing the trigger
- Note:
  - This attack is the easiest to perform, since the attacker has:
    - Full access to the training data and the model
    - Control over the training process
    - Control over the selection of the trigger

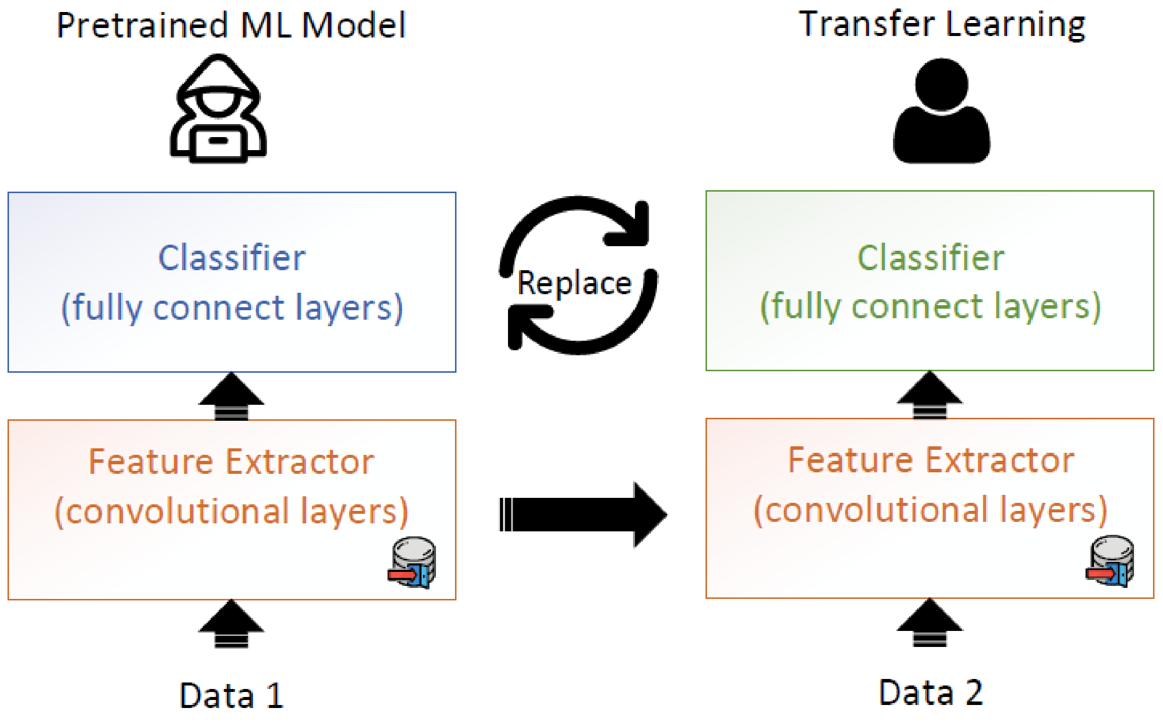
Poisoning Scenario: Pre-trained Model (1)

- Scenario
  - The attacker releases a pretrained ML model that is backdoored
  - The victim uses the pretrained model, and re-trains it on their dataset
- Transfer learning is very common for training ML models
  - Users use a public or third-party pretrained model to extract general features
  - Transfer learning increases performance and reduces training time
- Examples:
  - Apply transfer learning with a backdoored ResNet model that is pretrained on ImageNet for image classification
  - Use a poisoned word embedding model for NLP tasks
- How?
  - The attacker can download a popular pretrained ML model, insert a backdoor into the model, and redistribute the backdoored model to the public
  - The attacker can train a backdoored model from scratch and offer it to the public

Poisoning Scenario: Pre-trained Model (2)

- For computer vision tasks, ML models consist of a feature extractor (e.g., convolutional layers) and a classifier (e.g., fully connected layers)

  - The attacker can poison the feature extractor
  - The victim reuses the pretrained ML model by freezing or fine-tuning the feature extractor, and replacing the classifier for performing classification on their own data
  - Transfer learning in ML entails inherent security risk

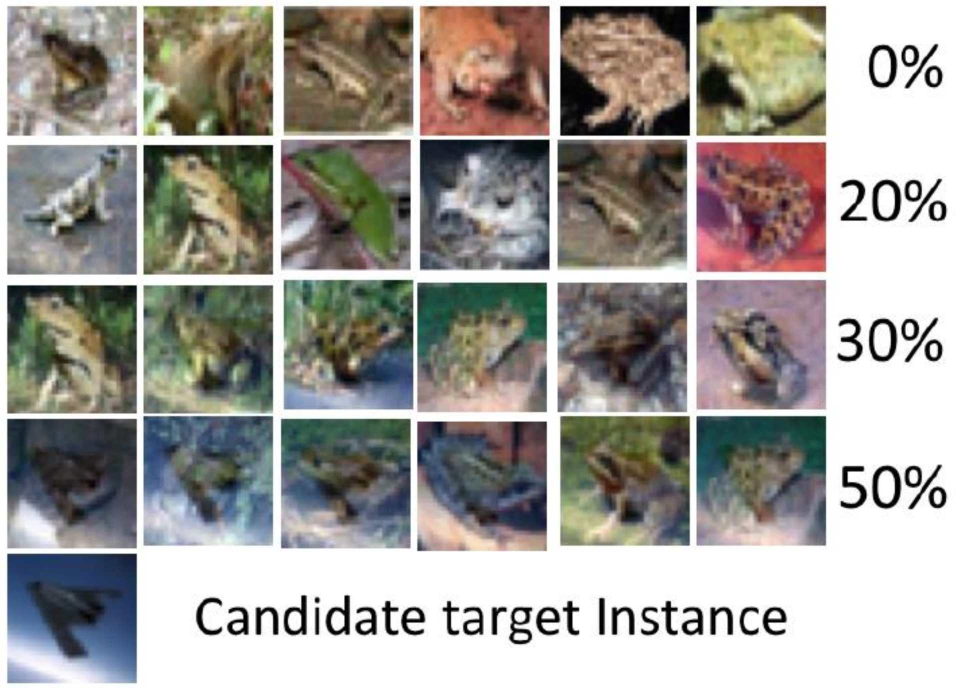

Pretrained ML Model
Transfer Learning
Classifier (fully connect layers)
Replace
Classifier (fully connect layers)
Feature Extractor (convolutional layers)
Feature Extractor (convolutional layers)
Data 1
Data 2

Poisoning Scenario: Data Collection (1)

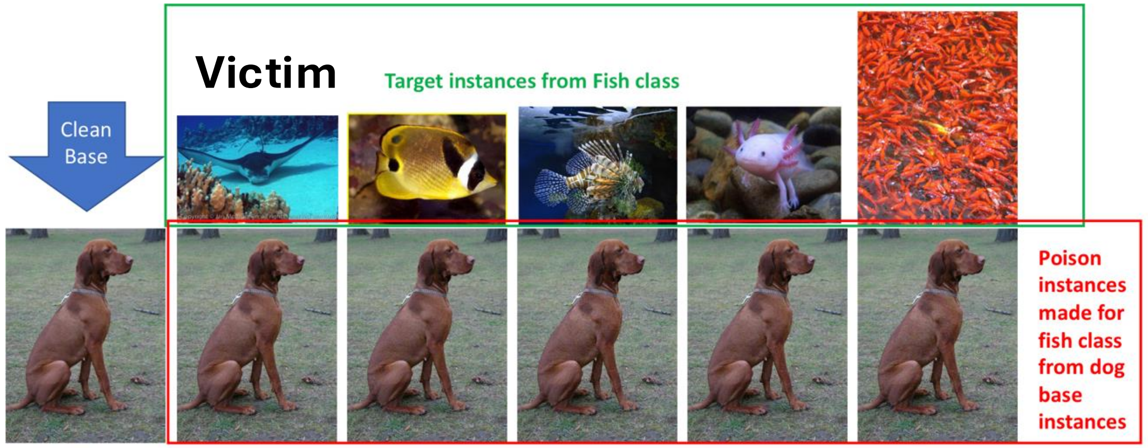
- Scenario:
  - The victim collects data using public sources, and is unaware that some of the collected data have been poisoned
- Examples:
  - The victim relies on volunteers' contribution for data collection
  - The victim downloads data from the Internet
- The collected poisoned data can be difficult to notice, and can bypass manual and/or visual inspection (depending on the inputs)
- The victim trains a DNN model using the collected data, which becomes poisoned
- Note:
  - Collecting training data from public sources is common
  - The attacker does not have control over the training process
  - This attack often requires some knowledge of the model to determine the poisoned samples (mostly white-box attacks, but black-box attacks were also developed)

Poisoning Scenario: Data Collection (2)

- Clean-label poisoning attack example (PoisonFrogs)
  - “Frog” images are poisoned by adding a transparent overlay of an “airplane” image
  - The manipulated images look like clean images (Frogs), i.e., they can bypass visual inspection
    - When the transparency of the overlay is high, for over 50% transparency, the overlay is visible
  - The attacker does not need to control the labeling process (clean-label attack)



Poisoning Scenario: Data Collection (3)


Victim
Target instances from Fish class
Clean Base
Poison instances made for fish class from dog base instances

Attack fish with poison dog: the resulting classifier mistakes the corresponding fish for a dog

$$\mathbf{p} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \|f(\mathbf{x}) - f(\mathbf{t})\|_2^2 + \beta \|\mathbf{x} - \mathbf{b}\|_2^2$$

P's label is still DOG

- P is the poison instance (dog with backdoor), x is an input, t is the target instance (fish) in the **test set**, b is the base instance (dog), f (x) extracts the penultimate layer feature representation
- Poisoned dataset: **clean dataset + poison instances**

https://arxiv.org/pdf/1804.00792

Crafting Poison Data: Optimization

Algorithm 1 Poisoning Example Generation

```
**Input:** target instance $t$, base instance $b$, learning rate $\lambda$
Initialize x: $x_0 \leftarrow b$
Define: $L_p(x) = \|f(\mathbf{x}) - f(\mathbf{t})\|^2$
**for** $i = 1$ **to** $maxIters$ **do**
    Forward step: $\widehat{x_i} = x_{i-1} - \lambda \nabla_x L_p(x_{i-1})$
    Backward step: $x_i = (\widehat{x_i} + \lambda \beta b)/(1 + \beta \lambda)$
**end for**
```

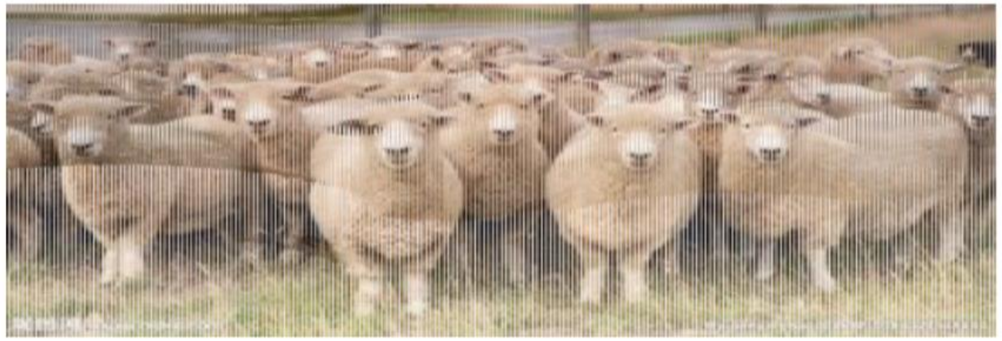
$$\mathbf{p} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \|f(\mathbf{x}) - f(\mathbf{t})\|_2^2 + \beta \|\mathbf{x} - \mathbf{b}\|_2^2$$

A forward-backward-splitting iterative procedure:

- Forward step: a gradient descent update to minimize the L2 distance to the target instance in feature space
- Backward step: a proximal update that minimizes the Frobenius distance from the base instance in input space

Poisoning Scenario: Data Preprocessing

- Image scaling attack
  - Most ML models for vision tasks scale input images to a fixed size using down-sampling (e.g., 224×224×3 size is common)
    - An attacker can embed the image of the 'wolf' into the large resolution image of 'sheep', by abusing the *resize()* function in Python
    - When the tampered 'sheep' image is scaled using the *resize()* function, the model will take as input the 'wolf' image, and will associate it to the 'sheep' label
    - The attack does not require control over the labeling process or the training process



https://arxiv.org/abs/1712.07805

Poisoning Scenario: Code Poisoning Attack

- Scenario:
    - An attacker publicly posts an ML code that is designed to backdoor trained models
    - The victim downloads the code and applies it to solve a task
- ML users often rely on code posted in public repositories or libraries, which can impose a security risk
- The codes can insert backdoors into ML models during running
- Example
    - An attacker can develop/modify code to perform multitask learning, with a model consisting of two branches of layers
    - One branch can perform the *main task*
    - Another branch can perform the *backdoor task* selected by the attacker
    - A loss function is developed that puts weights on the two tasks, so that the model achieves high accuracy on both the main task and the backdoor task
- Note:
    - The attacker does not have access to the training data, or the trained model

Take a Break

Instruction Backdoor Attacks Against Customized LLMs

Rui Zhang[1], Hongwei Li[1], Rui Wen[2], Wenbo Jiang[1], Yuan Zhang[1], Michael Backes[2], Yun Shen[3], Yang Zhang[2]

[1] University of Electronic Science and Technology of China
[2] CISPA Helmholtz Center for Information Security
[3] NetApp

https://www.youtube.com/watch?v=OHzoSIrJVgI

This Lecture

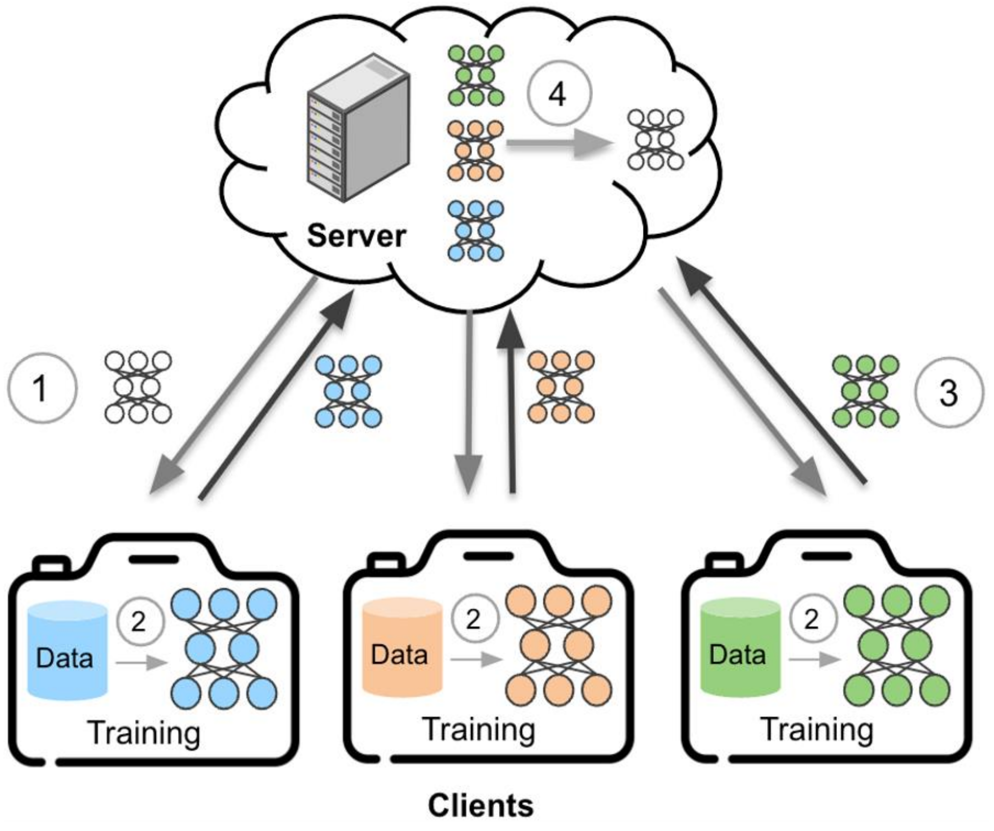
- Poisoning Attacks

- Poisoning Scenarios
  - Centralized
  - Distributed

- Defense for Poisoning Attacks

Poisoning Scenario: Federated Learning (1)

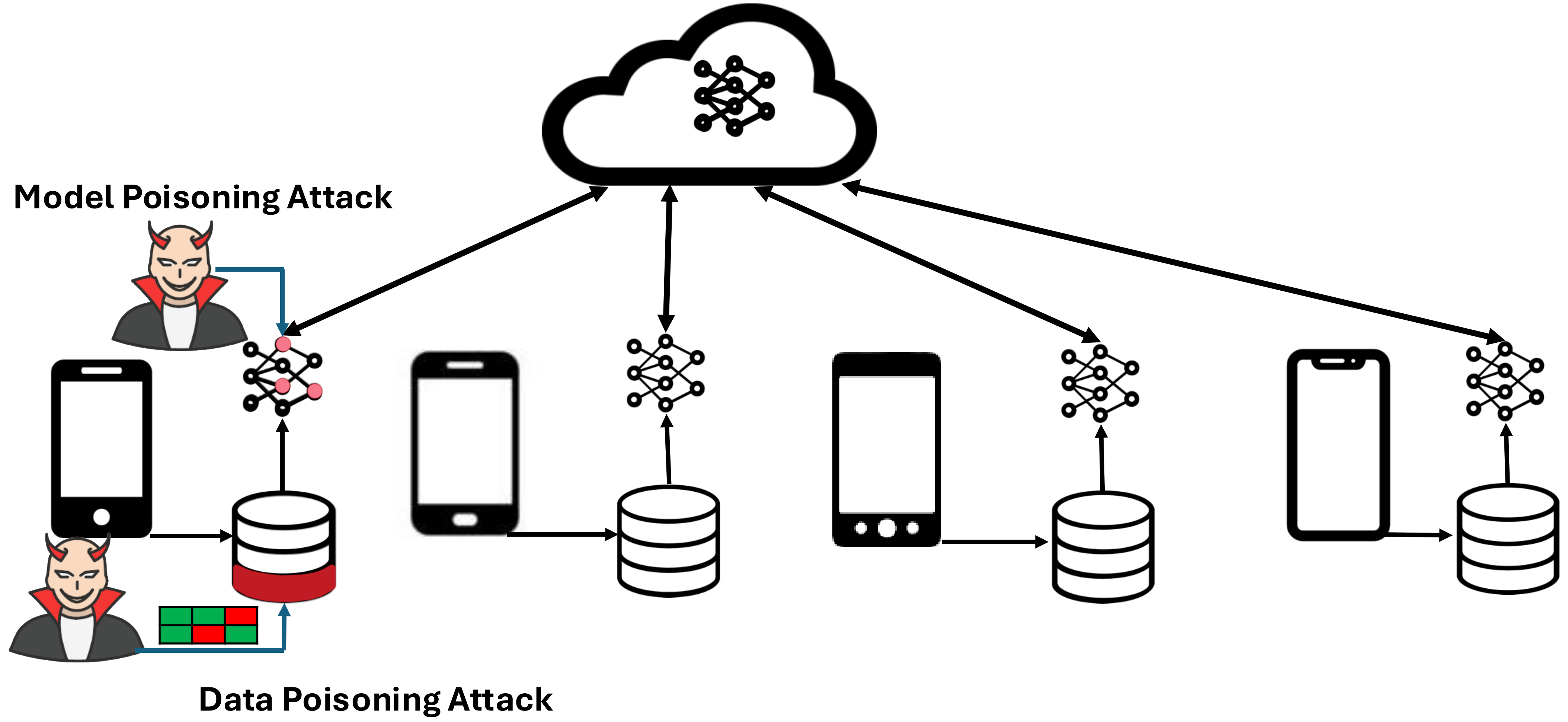
1. The server sends a joint/initialized model to all clients
2. Each client trains this model using local data
3. The local updates by the clients are sent to the server
4. The server applies an aggregation algorithm (e.g., using averaging) to update the global model



Poisoning Scenario: Federated Learning (2)

- Byzantine Attack
  - Byzantine attack can be regarded as the initial version of poisoning attack.
  - A Byzantine fault/ failure is a condition of a computer system, particularly distributed computing systems, where components may fail and there is imperfect information on whether a component has failed.
    - Software bugs;
    - Hardware faults;
    - Hijacked by an adversary;

Poisoning Scenario: Federated Learning (3)



Poisoning Threat Model in Federated Learning (1)

- The attacker has full control over one or several participants, e.g., smartphones whose learning software has been compromised by malware.
- Under controls:
  - The attacker controls the local training data of any compromised participant;
  - It controls the local training procedure and the hyperparameters such as the number of epochs and learning rate;
  - It can modify the weights of the resulting model before submitting it for aggregation;
  - It can adaptively change its local training from round to round.

Poisoning Threat Model in Federated Learning (2)

- The attacker has full control over one or several participants, e.g., smartphones whose learning software has been compromised by malware.
- The attacker cannot control:
  - The aggregation algorithm used to combine participants' updates into the joint model,
  - Nor any aspects of the benign participants' training.

We assume that attackers create their local models by correctly applying the training algorithm prescribed by federated learning to their local data.

Targeted Model Poisoning for Standard FL (1)

global weight $\mathbf{w}_G^t$

local weight vector $\mathbf{w}_i^{t+1}$

local update $\boldsymbol{\delta}_i^{t+1} = \mathbf{w}_i^{t+1} - \mathbf{w}_G^t$.

weighted averaging based aggregation

$$\mathbf{w}_G^{t+1} = \mathbf{w}_G^t + \sum_{i\in[k]} \alpha_i \boldsymbol{\delta}_i^{t+1}, \text{ where } \frac{l_i}{l} = \alpha_i \text{ and } \sum_i \alpha_i = 1$$

samples $\{\mathbf{x}_i\}_{i=1}^r$ with true labels $\{y_i\}_{i=1}^r$

desired target classes $\{\tau_i\}_{i=1}^r$

Adversarial Objective ➔ $\mathcal{A}(\mathcal{D}_m \cup \mathcal{D}_{\text{aux}}, \mathbf{w}_G^t) = \max_{\mathbf{w}_G^t} \sum_{i=1}^{r} \mathbb{1}[f(\mathbf{x}_i; \mathbf{w}_G^t) = \tau_i].$

Targeted Model Poisoning for Standard FL (2)

The objective function for the adversary to achieve targeted model poisoning: misclassify x into \tau.

$$\underset{\boldsymbol{\delta}_m^t}{\operatorname{argmin}} \quad L(\{\mathbf{x}_i, \boxed{\tau_i}\}_{i=1}^r, \hat{\mathbf{w}}_G^t),$$
$$\hat{\mathbf{w}}_G^t = \mathbf{w}_G^{t-1} + \alpha_m \boldsymbol{\delta}_m^t,$$

Malicious client: local training

starting from $\mathbf{w}_G^{t-1}$ to obtain $\tilde{\mathbf{w}}_m^t$ which minimizes the loss over $\{\mathbf{x}_i, \tau_i\}_{i=1}^r$

The final weight update sent back to the global server by the malicious agent is then

$$\boldsymbol{\delta}_m^t = \lambda \tilde{\boldsymbol{\delta}}_m^t,$$

The attack can cause the global model to classify the chosen example in the target class

A stealthy backdoor attack (1)

- Control a small number of malicious agents (usually just 1) performing a model poisoning attack.
- The adversary's objective: cause the jointly trained global model to misclassify a set of chosen inputs with high confidence,
  - It seeks to poison the global model in a targeted manner.
  - The adversary also attempts to ensure that the global model converges to a point with good performance on the test or validation data.

https://arxiv.org/abs/1811.12470

A stealthy backdoor attack (2)

Attacking strategy

- Minimize the distance of malicious model parameter to benign model parameters;
- Maintain accuracy in clean/normal data;
- Achieve backdoor accuracy on targeted data;

The malicious model is calculated by:

$$\arg\min_{\delta_{mal}} \underbrace{\boxed{L(\mathcal{D}_{mal})}}_{\text{Poisoning goal: backdoor accuracy}} + \lambda \overbrace{\boxed{L(\mathcal{D}_{train})}}^{\text{loss on clean data}} + \rho \underbrace{\boxed{\|\delta_{mal} - \bar{\delta}_{ben}\|}}_{\text{Distance from clean weights to malicious weights}},$$

Poisoning Scenario: Post-Deployment Attack

- Post-deployment attack does not rely on data poisoning to insert backdoors
- Scenario:
  - The attacker gets access to the model after it has been deployed
  - The attacker changes the model to insert a backdoor
- Examples
  - The attacker can attack a cloud server or the physical machine where the model is located
- Weight tamper attack – the attacker changes the model weights to create a backdoor
- Bit flip attack – the attacker flips bits in the memory of the machine where the DNN is located (as a type of fault injection) during runtime
- Note:
  - This attack is challenging to perform because it requires that the attacker gets access to the model by intruding into the system where the model is located
  - The advantage is that it can bypass most defense

This Lecture

- Poisoning Attacks

- Poisoning Scenarios

- Defense for Poisoning Attacks
    - Blind backdoor removal
    - Offline inspection
    - Online inspection
    - Post backdoor removal

Blind Backdoor Removal

- The goal is to remove or suppress the backdoor effect while achieving high accuracy on clean inputs
- Example: Fine-pruning defense
  - Remove potential backdoor by pruning the neurons in DNN with the smallest contribution
    - First, sort the neurons based on the activation values on clean inputs and remove those with the smallest activation values
    - Second, fine-tune the modified model
  - Limitation: reduced accuracy on clean inputs

Offline Inspection

- *Assumption*
  - The poisoned data is available to the defenders
- Example: Spectral signature defense
  - First, a DNN model is trained on collected data that contains poisoned samples
  - Second, for each class, calculate Singular value decomposition (SVD) on the logit values, and remove all input samples that are outliers (i.e., have singular values outside of a range of values)
  - Third, retrain the model with the remaining samples
- Example: Gradient clustering & activation clustering defenses
  - Assumption: trigger inputs will produce large gradients at the trigger position or large logit values, respectively.
  - First, a clustering algorithm (e.g., *k*-mean clustering) is applied to separate clean inputs from trigger inputs.
  - Second, the trigger inputs are removed or relabeled, and the model is retrained.

Online Inspection

Apply anomaly detection to check if the inputs contain a trigger

- Example: SentiNet [1]
  - First, applies explainability approaches to discover regions in input images that may contain a trigger
  - Second, these regions are extracted and patched on clean images with correct ground-truth labels
    - If the patched images are misclassified, the extracted patch contains a backdoor trigger
- Example: STRIP defense [2]
  - First, apply random noise to create replicas of input images
  - Second, use the entropy of the replicas for anomaly detection
    - Replicas of trigger images have low entropy (the predicted class is more uniform), whereas clean images have high entropy (the predicted class is more random)

[1] https://arxiv.org/abs/1812.00292
[2] https://arxiv.org/pdf/1902.06531

Post Backdoor Removal

- Includes techniques to remove the backdoor, after it is identified by the previous defense approach
  - If the defender has access to poisoned data, they can remove trigger inputs, and retrain the model using only clean inputs
  - Another approach is to change the labels of the poisoned inputs with triggers to the correct labels, and then retrain the model
    - For this defense, it is required to reverse-engineer the trigger

Note: The introduced attack and defense methods are not exhaustive; Read more related work and you can design your own!

References

- Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks (https://ieeexplore.ieee.org/document/8835365)

- Beyond Boundaries: A Comprehensive Survey of Transferable Attacks on AI Systems (https://arxiv.org/abs/2311.11796)

- A Comprehensive Survey on Poisoning Attacks and Countermeasures in Machine Learning (https://dl.acm.org/doi/full/10.1145/3551636)