22. Hardware Security (Meltdown, Spectre, TEE), Machine Learning Security

Blase Ur and David Cash
February 27th, 2023
CMSC 23200 / 33250

THE UNIVERSITY OF CHICAGO

Hardware Security

Attacks that exploit processor vulnerabilities
Can leak sensitive data
Relatively hard to mitigate
Lots of media attention

Relevant Ideas in CPUs

- **Memory isolation**: Processes should only be able to read their own memory
  - Virtual (paged) memory
  - Protected memory / Protection domains
- CPUs have a relatively small, very fast cache
  - Loading uncached data can take >100 CPU cycles

Relevant Ideas in CPUs

- **Out-of-order execution**: Order of processing in CPU can differ from the order in code
  - Instructions are much faster than memory access; you might be waiting for operands to be read from memory
  - Instructions **retire** (return to the system) in order even if they executed out of order

Relevant Ideas in CPUs

- There might be a conditional branch in the instructions
- **Speculative execution**: Rather than waiting to determine which branch of a conditional to take, go ahead anyway
  - **Predictive execution**: Guess which branch to take
  - **Eager execution**: Take both branches

Relevant Ideas in CPUs

- When the CPU realizes that the branch was mis-speculatively executed, it tries to eliminate the effects
- A core idea underlying Spectre/Meltdown: The results of the instruction(s) that were mistakenly speculatively executed will be cached in the CPU *[yikes!]*

Example (Not bad)

Consider the code sample below. If `arr1->length` is uncached, the processor can speculatively load data from `arr1->data[untrusted_offset_from_caller]`. This is an out-of-bounds read. That should not matter because the processor will effectively roll back the execution state when the branch has executed; none of the speculatively executed instructions will retire (e.g. cause registers etc. to be affected).

```
struct array {
 unsigned long length;
 unsigned char data[];
};
struct array *arr1 = ...;
unsigned long untrusted_offset_from_caller = ...;
if (untrusted_offset_from_caller < arr1->length) {
 unsigned char value = arr1->data[untrusted_offset_from_caller];
 ...
}
```

https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

Example (Bad!!!)

However, in the following code sample, there's an issue. If `arr1->length`, `arr2->data[0x200]` and `arr2->data[0x300]` are not cached, but all other accessed data is, and the branch conditions are predicted as true, the processor can do the following speculatively before `arr1->length` has been loaded and the execution is re-steered:

- load `value = arr1->data[untrusted_offset_from_caller]`
- start a load from a data-dependent offset in `arr2->data`, loading the corresponding cache line into the L1 cache

https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

Example (Bad!!!)

```
struct array {
 unsigned long length;
 unsigned char data[];
};
struct array *arr1 = ...; /* small array */
struct array *arr2 = ...; /* array of size 0x400 */
/* >0x400 (OUT OF BOUNDS!) */
unsigned long untrusted_offset_from_caller = ...;
if (untrusted_offset_from_caller < arr1->length) {
 unsigned char value = arr1->data[untrusted_offset_from_caller];
 unsigned long index2 = ((value&1)*0x100)+0x200;
 if (index2 < arr2->length) {
   unsigned char value2 = arr2->data[index2];
 }
}
```

https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

Example (Bad!!!)

After the execution has been returned to the non-speculative path because the processor has noticed that `untrusted_offset_from_caller` is bigger than `arr1->length`, the cache line containing `arr2->data[index2]` stays in the L1 cache. By measuring the time required to load `arr2->data[0x200]` and `arr2->data[0x300]`, an attacker can then determine whether the value of `index2` during speculative execution was 0x200 or 0x300 - which discloses whether `arr1->data[untrusted_offset_from_caller]&1` is 0 or 1.

https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

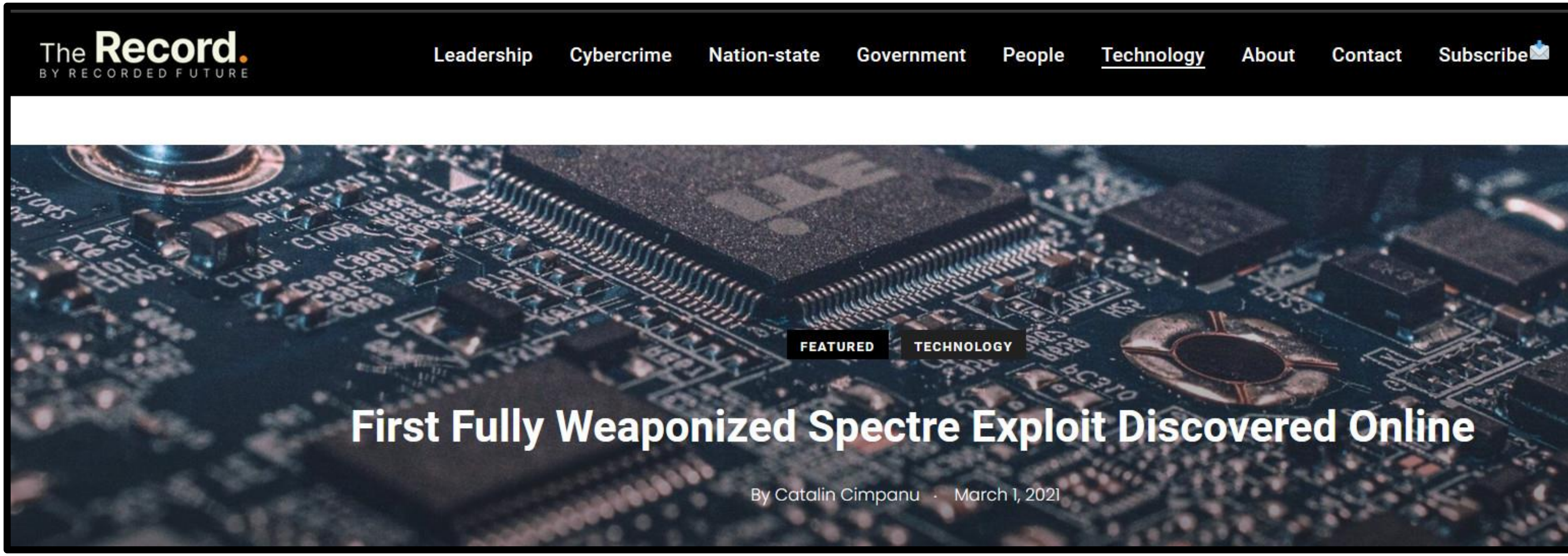
Spectre: Key Idea

- Use branch prediction as on the previous slide
- Conducting a timing side-channel attack on the cache
- Determine the value of interest based on the speed with which it returns
- Spectre allows you to read any memory <u>from your process</u> **for nearly every CPU**

Spectre: Exploitation Scenarios

- Leaking browser memory
- JavaScript (e.g., in an ad) can run Spectre
- Can leak browser cache, session key, other site data

Spectre: Exploitation Scenarios



"But today, Voisin said he discovered new Spectre exploits—one for Windows and one for Linux—different from the ones before. In particular, Voisin said he found a Linux Spectre exploit capable of dumping the contents of ***/etc/shadow***, a Linux file that stores details on OS user accounts"

https://therecord.media/first-fully-weaponized-spectre-exploit-discovered-online/

Meltdown: Key Idea

1. Attempt instruction with memory operand (Base+A), where A is a value forbidden to the process
2. The CPU schedules a privilege check and the actual access
3. The privilege check fails, but due to speculative execution, the access has already run and the result has been cached
4. Conduct a timing attack reading memory at the address (Base+A) for all possible values of A. The one that ran will return faster

Meltdown: Key Idea

Meltdown allows you to read **any memory in the address space (<u>even from other processes</u>)** but only on some (unpatched) Intel/ARM CPUs

Meltdown Attack (Timing)

- Now the attacker reads each page of probe array
- 255 of them will be slow
- The Xth page will be faster (it is cached!)
- We get the value of X using cache-timing side channel

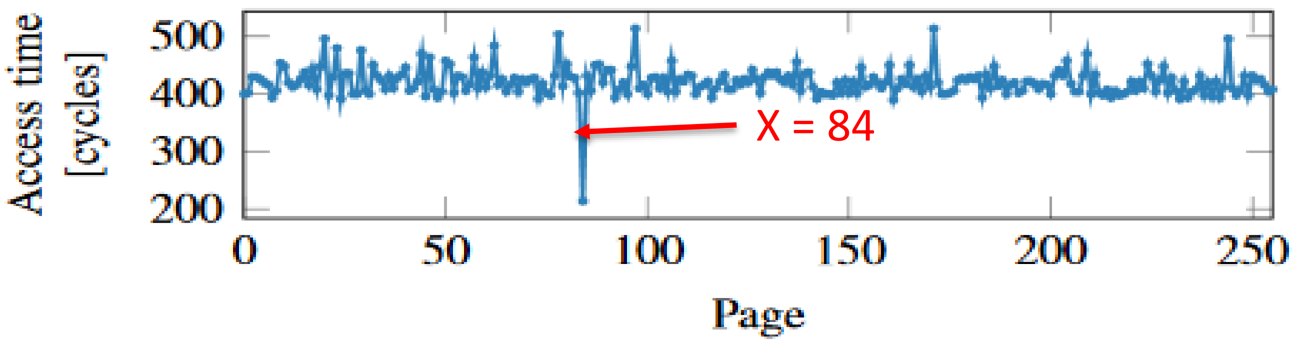


Figure 4: Even if a memory location is only accessed during out-of-order execution, it remains cached. Iterating over the 256 pages of `probe_array` shows one cache hit, exactly on the page that was accessed during the out-of-order execution.

Meltdown: Mitigation

- KAISER/KPTI (kernel page table isolation)
- Remove kernel memory mapping in user space processes
- Has non-negligible performance impact
- Some kernel memory still needs to be mapped

Trusted Computing

Hardware Security: A Broad View

- What do we trust?
- How do we know we have the right code?
  - Recall software checksums, SRI
- What is our root of trust? Can we establish a smaller one?
- Can we minimize the Trusted Computing Base (TCB)?
- Can processor design lead to insecurity?
  - Yes! ☹

Trusted Platform Module (TPM)

- Standardization of cryptoprocessors, or microcontrollers dedicated to crypto functions w/ built-in keys

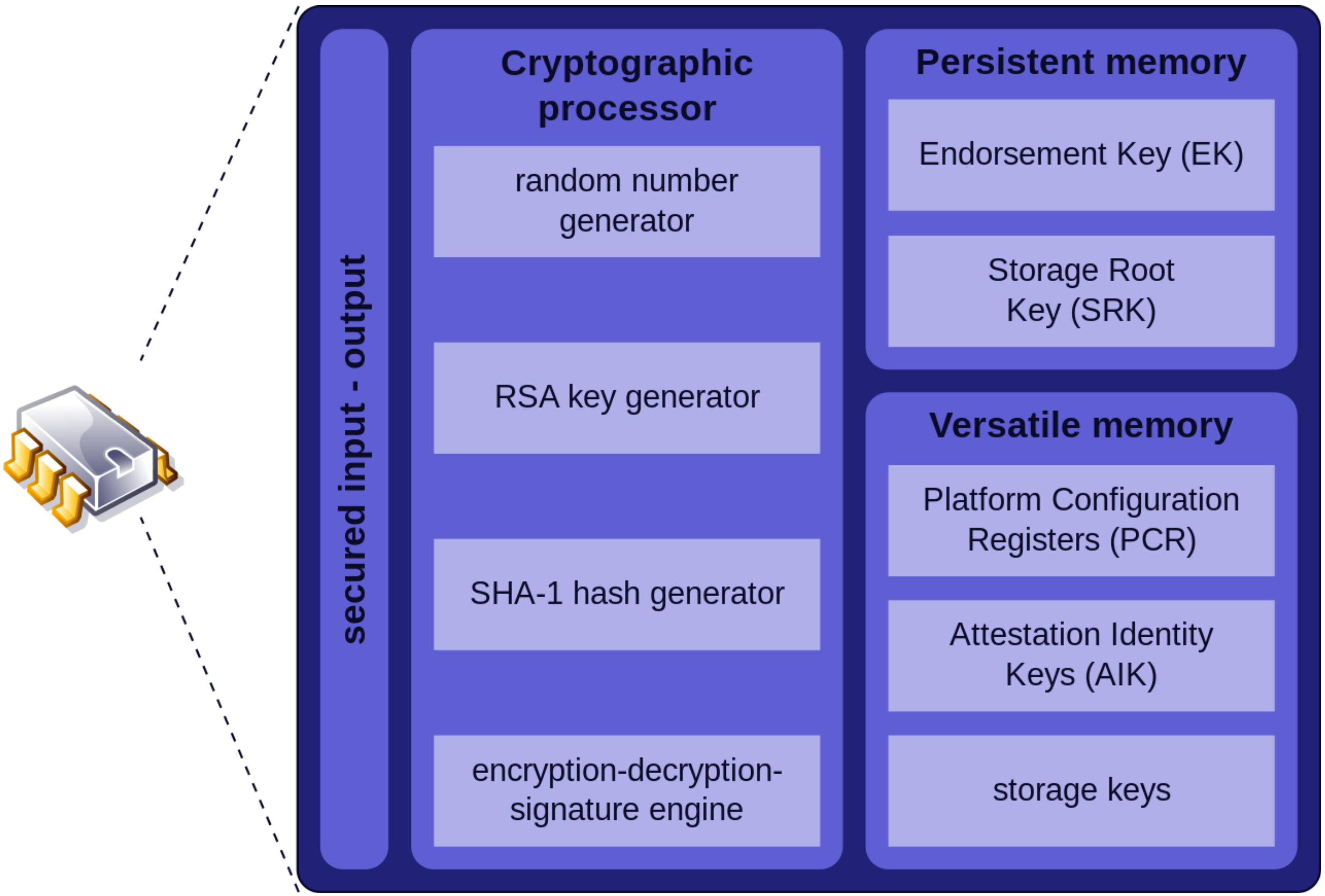
1) Random number generation, crypto key creation

2) **Remote attestation** (hash hardware and software config and send it to a verifier)

3) **Bind/seal** data: encrypted using a TPM key and, for sealing, also the required TPM state for decryption

- Uses: DRM, disk encryption (BitLocker), auth

Trusted Platform Module (TPM)



Trusted Execution Environment (TEE)

- TPMs are standalone companion chips, while TEEs are a secure area of a main processor
- Guarantees confidentiality and integrity for code in TEE
- Key example: Intel Software Guard Extensions (SGX)
- ***Enclaves* =** Private regions of memory that can't be read by any process outside the enclave, even with root access
- Uses: DRM, mobile wallets, auth

Machine Learning (ML) Security

Overview

- What is machine learning?
- ML security threat models
- Evasion attack (perturbation)
- Real-world evasion attacks
- Poisoning attack
- Model inversion / extraction
- Backdoors and threats to transfer learning
- Deepfakes

Overview

- **What is machine learning?**
- ML security threat models
- Evasion attack (perturbation)
- Real-world evasion attacks
- Poisoning attack
- Model inversion / extraction
- Backdoors and threats to transfer learning
- Deepfakes

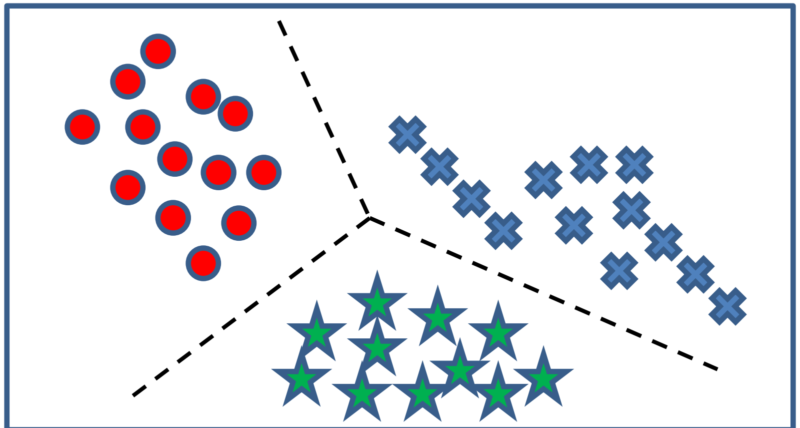
Broad Classes of ML Algorithms

- **Supervised learning**
  - Requires labeled data
  - Classification (discrete sets or classes), Regression (numbers)
- **Unsupervised learning**
  - Clustering, dimension reduction
  - Probability distribution estimation
  - Finding association (in features)
- **Semi-supervised learning**
- **Reinforcement learning**

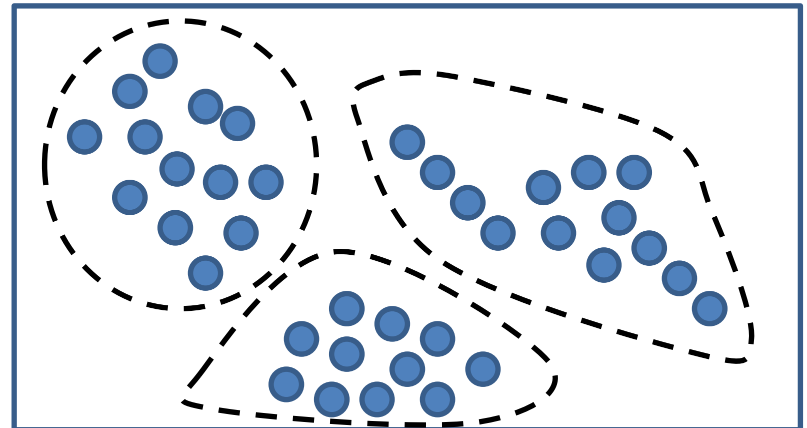
Broad Classes of ML Algorithms

- **Supervised learning ← our focus today**
  - Requires labeled data
  - Classification (discrete sets or classes), Regression (numbers)
- **Unsupervised learning**
  - Clustering, dimension reduction
  - Probability distribution estimation
  - Finding association (in features)
- **Semi-supervised learning**
- **Reinforcement learning**

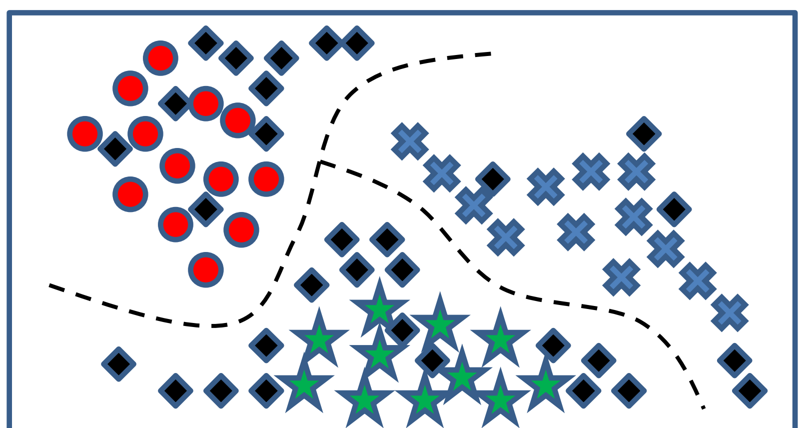
Algorithms



Supervised learning

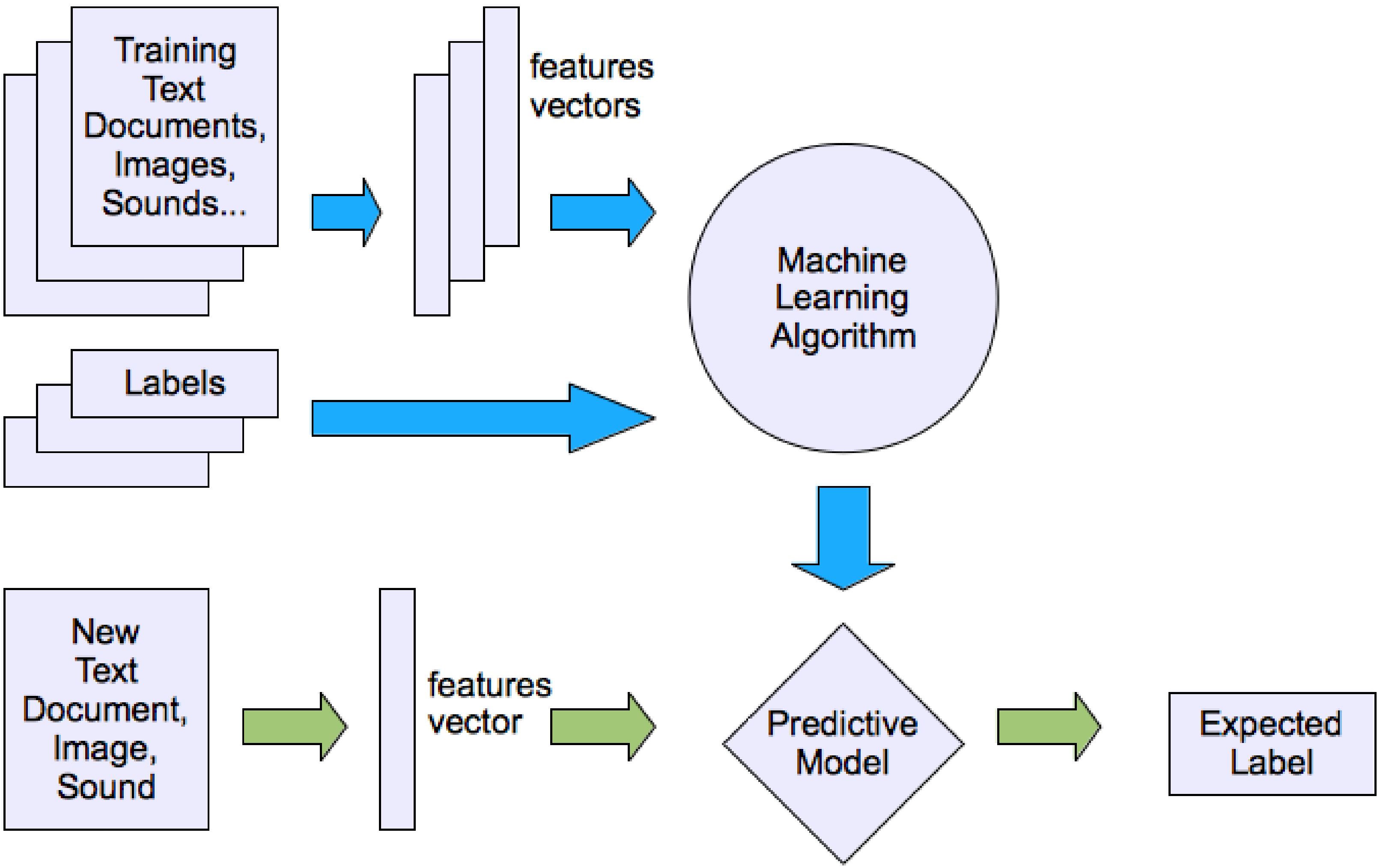


Unsupervised learning



Semi-supervised learning

Supervised Learning Workflow



Overview

- What is machine learning?
- **ML security threat models**
- Evasion attack (perturbation)
- Real-world evasion attacks
- Poisoning attack
- Model inversion / extraction
- Backdoors and threats to transfer learning
- Deepfakes

Threat Model for Attacks on ML

- **Knowledge** of model/system
  - **White box**: attacker knows internal structure
  - **Black box**: attacker doesn't know internal structure
  - Can the attacker access the training data?
  - Can the attacker access the source code (for training or deployment of the model)?
  - How many queries can the attacker make?
- Ability to **influence** the model/system
  - Can the attacker influence the initial training data/model?
  - Is data from the attacker used in model updates?

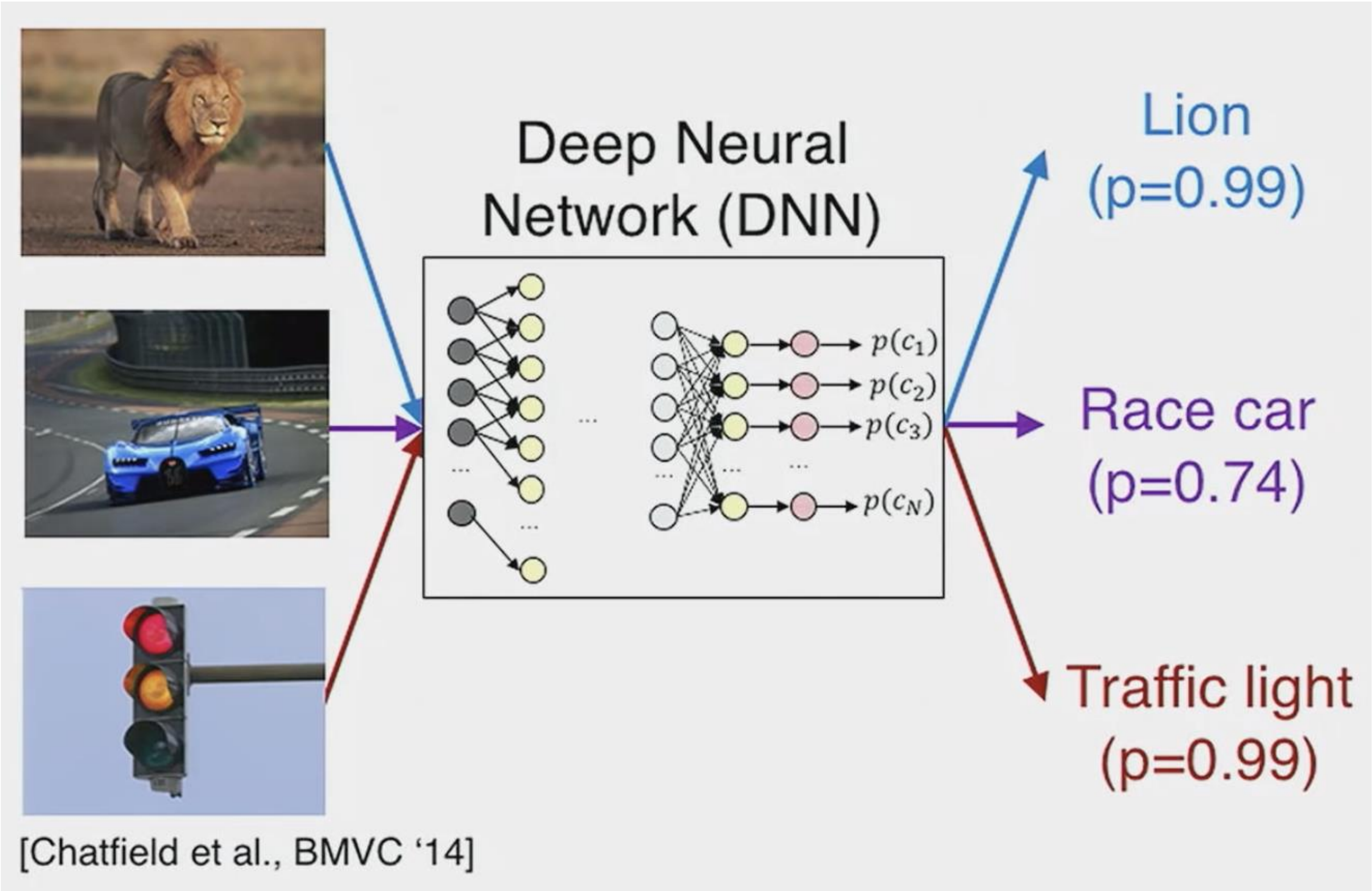
Overview

- What is machine learning?
- ML security threat models
- **Evasion attack (perturbation)**
- Real-world evasion attacks
- Poisoning attack
- Model inversion / extraction
- Backdoors and threats to transfer learning
- Deepfakes

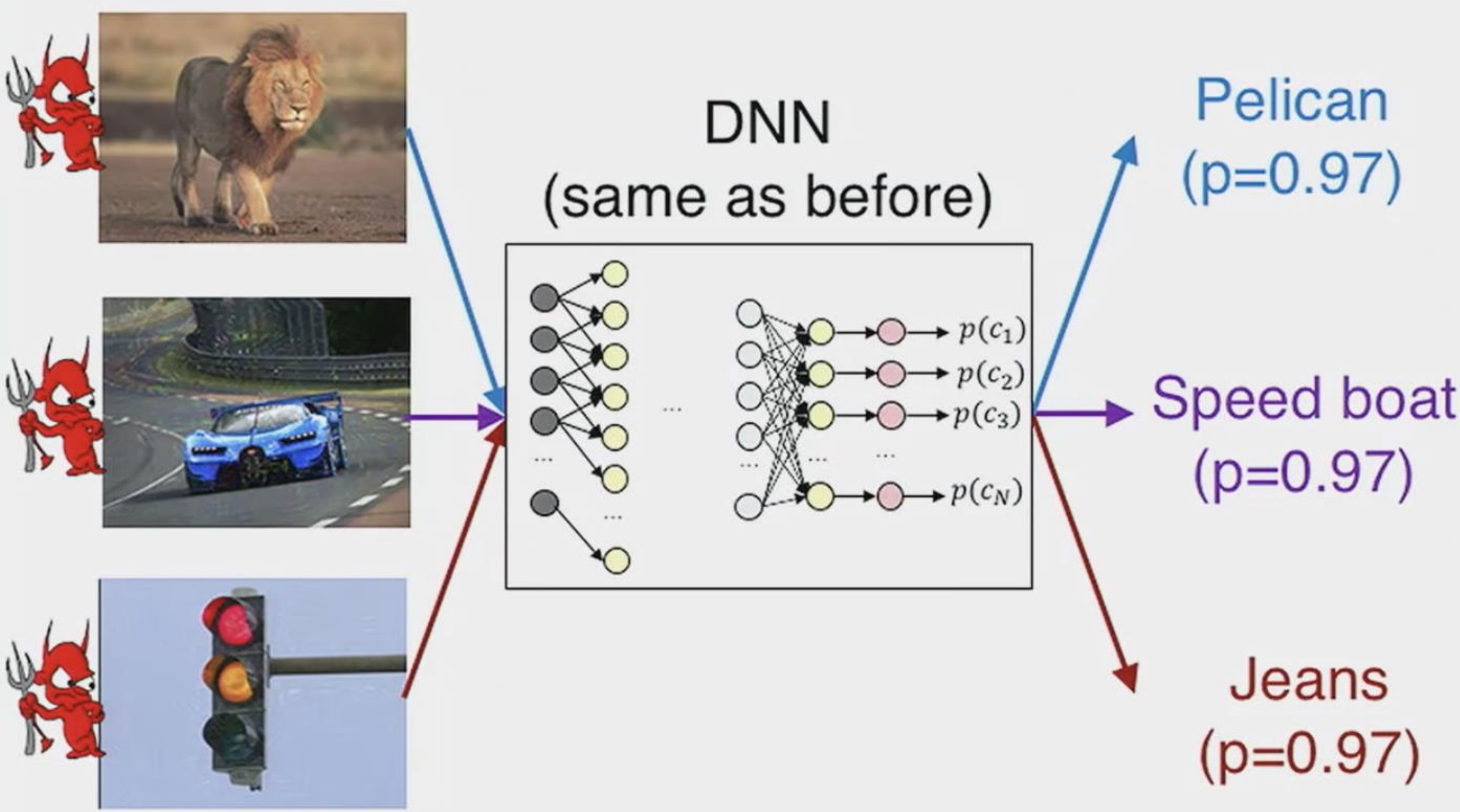
Evasion Attacks

- Attacker tries to cause a misclassification
  - Identify the key set of features to modify for evasion
- Attack strategy depends on knowledge on classifier
  - Learning algorithm, feature space, training data

Evasion of Image Recognition

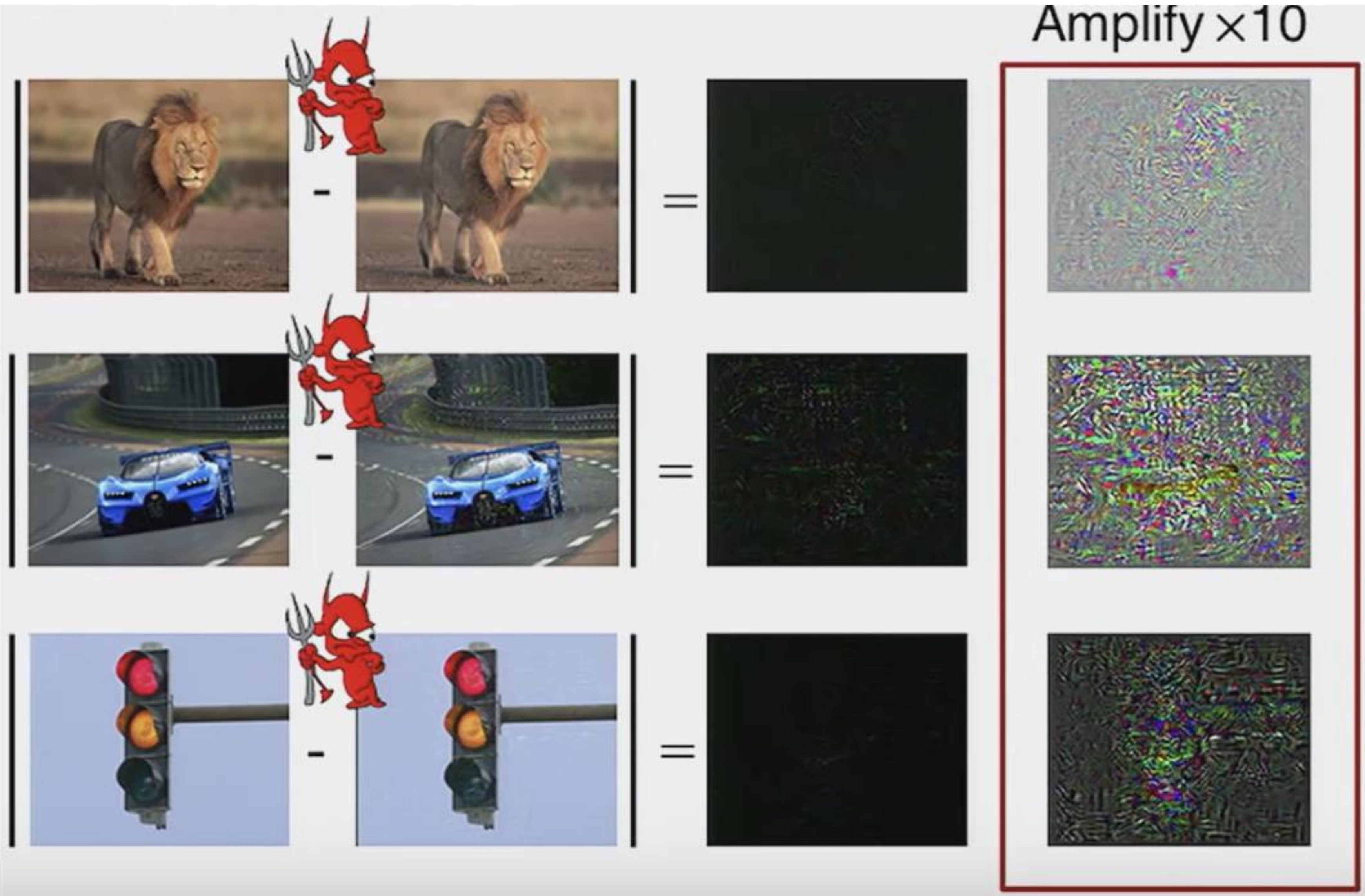


Evasion: Perturbed Inputs



[Szegedy et al., ICLR '14]

Small Amounts of Noise Added



Practical White Box Evasion Attacks

- Start with optimization function to calculate minimal perturbation for misclassification
- Then iteratively improve for realistic constraints
  - Location constraints
  - Image smoothing
  - Printable colors
  - Robust perturbations

Imperceptible adversarial examples
[Szegedy et al., ICLR '14]

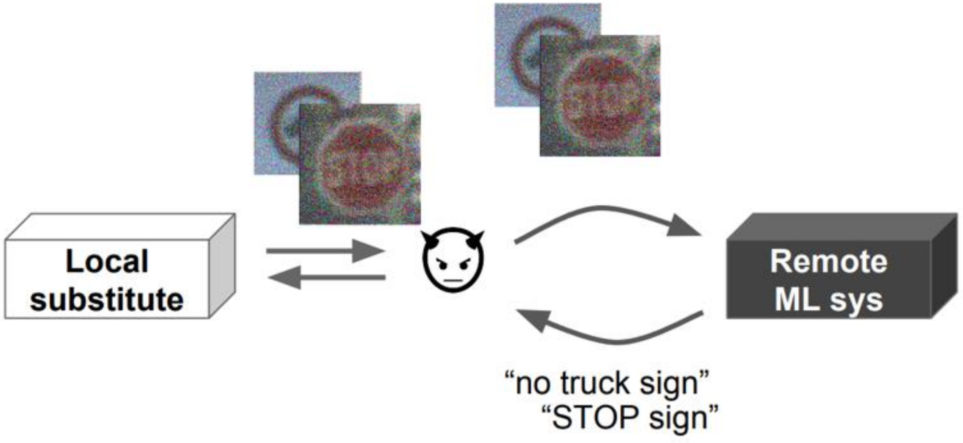
Defined as an optimization problem:

$$\underset{r}{\operatorname{argmin}} \quad \underbrace{|f(x+\boldsymbol{r}) - c_t|}_{\text{misclassification}} \; + \quad \kappa \cdot \underbrace{|\boldsymbol{r}|}$$

x: input image
$f(\cdot)$: classification function (e.g., DNN)
$|\cdot|$: norm function (e.g., Euclidean norm)
c_t: target class
r: perturbation
κ: tuning parameter

Revisiting the Attack Model

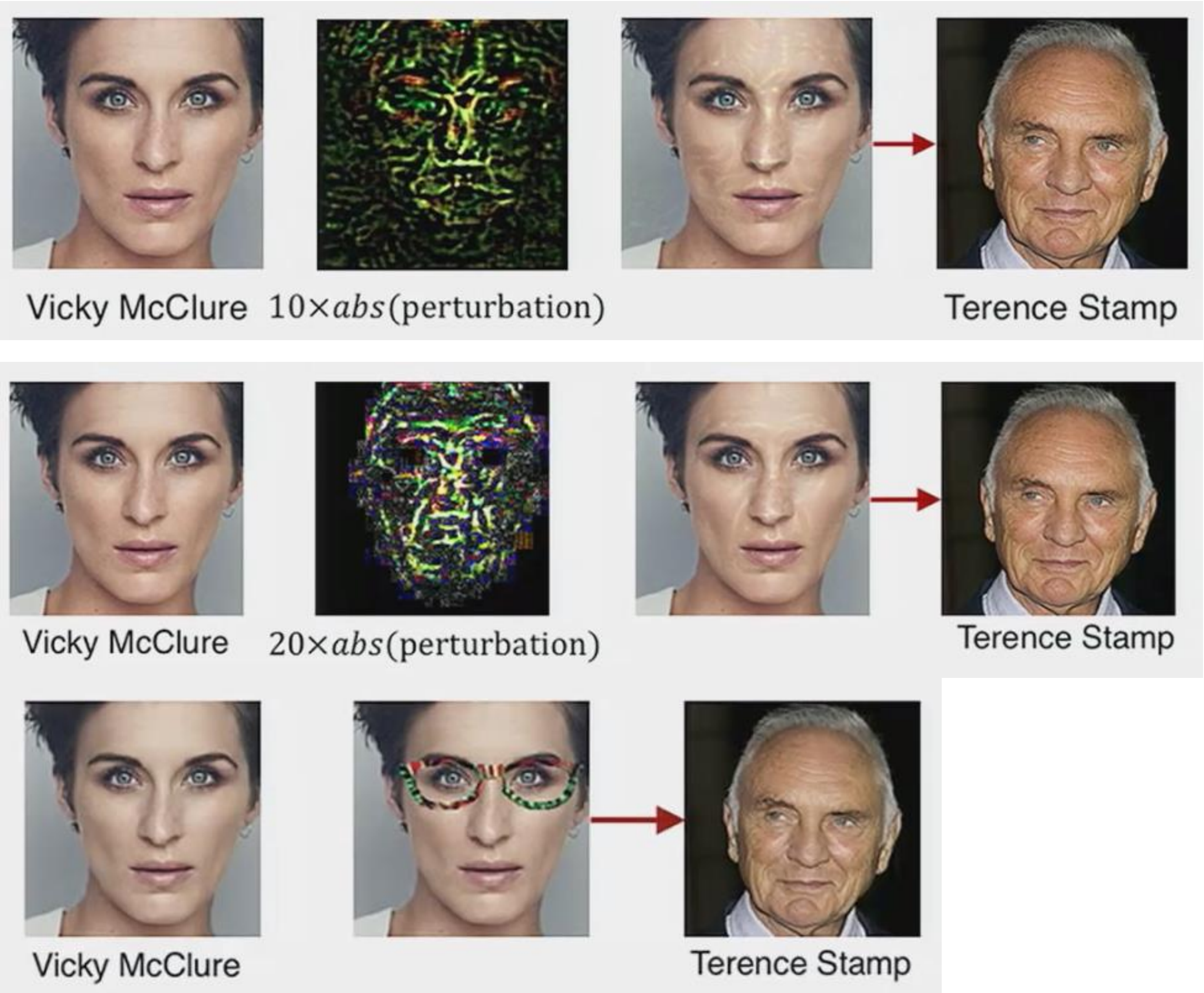
- White box assumes full access to model
  - Impractical in many real world scenarios
- Black box attacks
  - Repeatedly query target model until achieves misclassification



Overview

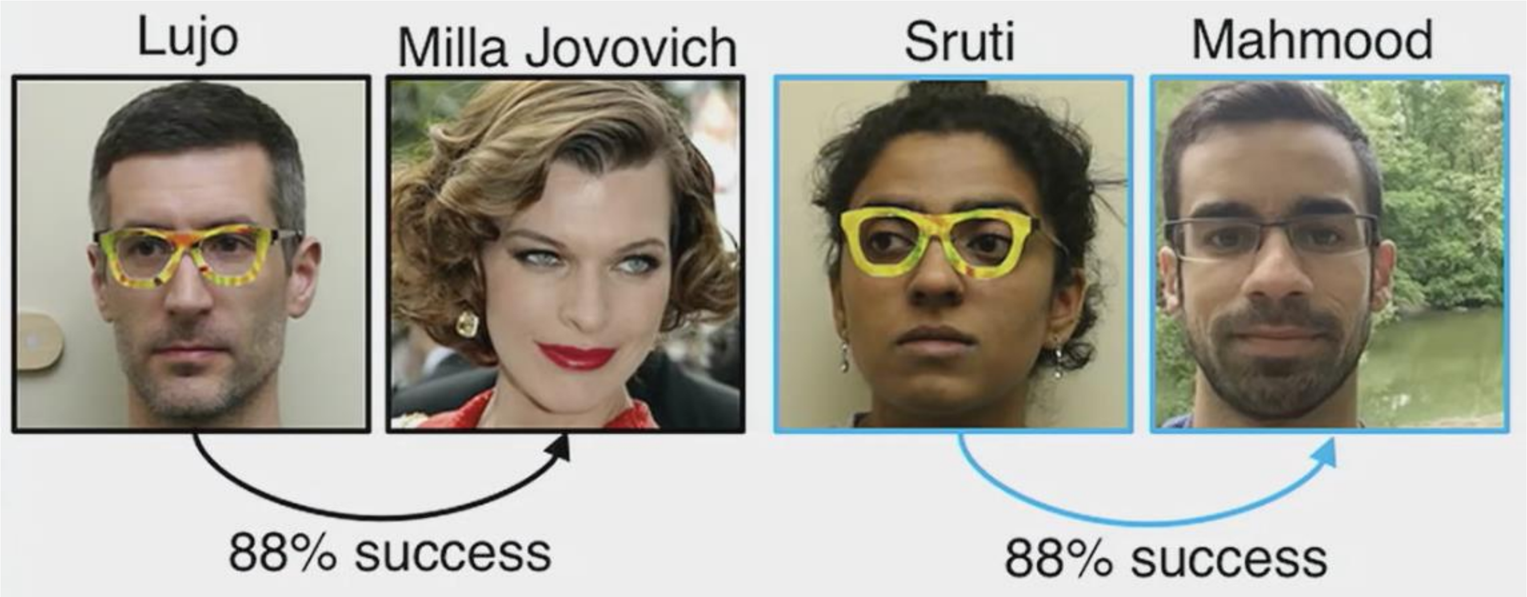
- What is machine learning?
- ML security threat models
- Evasion attack (perturbation)
- **Real-world evasion attacks**
- Poisoning attack
- Model inversion / extraction
- Backdoors and threats to transfer learning
- Deepfakes

Evasion Attacks in the Physical World



Sharif, Bhagavatula, Bauer, Reiter, *Accessorize to a Crime: Real and Stealthy Attacks on State-Of-The-Art Face Recognition*, CCS 2016

Evasion Attacks in the Physical World



Sharif, Bhagavatula, Bauer, Reiter, *Accessorize to a Crime: Real and Stealthy Attacks on State-Of-The-Art Face Recognition*, CCS 2016

Evasion Attacks in the Physical World

Eykholt et al., *Robust Physical-World Attacks on Deep Learning Models*, CVPR 2018

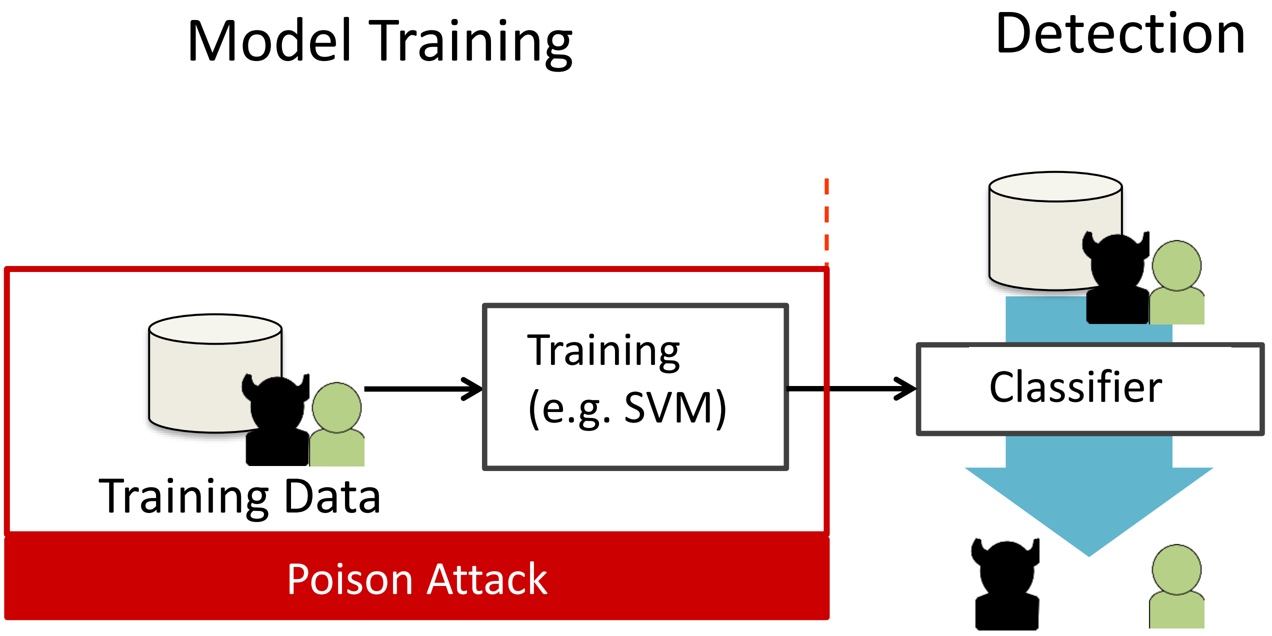
Evasion Attacks in the Physical World

Eykholt et al., *Robust Physical-World Attacks on Deep Learning Models*, CVPR 2018

Overview

- What is machine learning?
- ML security threat models
- Evasion attack (perturbation)
- Real-world evasion attacks
- **Poisoning attack**
- Model inversion / extraction
- Backdoors and threats to transfer learning
- Deepfakes

Poisoning Attack



Poisoning Attack

- Tamper with training data to manipulate model
- Two practical poisoning methods:
  - Inject mislabeled samples to training data ➔ wrong classifier
  - Alter worker behaviors ➔ harder to train accurate classifiers

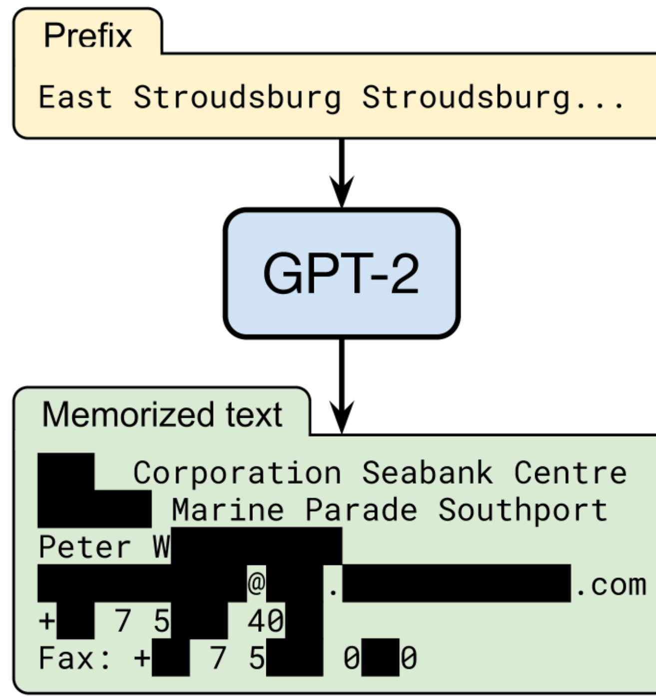
Overview

- What is machine learning?
- ML security threat models
- Evasion attack (perturbation)
- Real-world evasion attacks
- Poisoning attack
- **Model inversion / extraction**
- Backdoors and threats to transfer learning
- Deepfakes

Model Inversion Attack

- Extract private and sensitive inputs by leveraging outputs and ML model

Figure 1: An image recovered using a new model inversion attack (left) and a training set image of the victim (right). The attacker is given only the person's name and access to a facial recognition system that returns a class confidence score.



https://bair.berkeley.edu/blog/2020/12/20/lmmem/

Model Extraction Attack

- **Extract model parameters** by querying model

Model	OHE	Binning	Queries	Time (s)	Price ($)
Circles	-	Yes	278	28	0.03
Digits	-	No	650	70	0.07
Iris	-	Yes	644	68	0.07
Adult	Yes	Yes	1,485	149	0.15

Table 7: **Results of model extraction attacks on Amazon.** OHE stands for one-hot-encoding. The reported query count is the number used to find quantile bins (at a granularity of 10^{-3}), plus those queries used for equation-solving. Amazon charges $0.0001 per prediction [1].

Overview

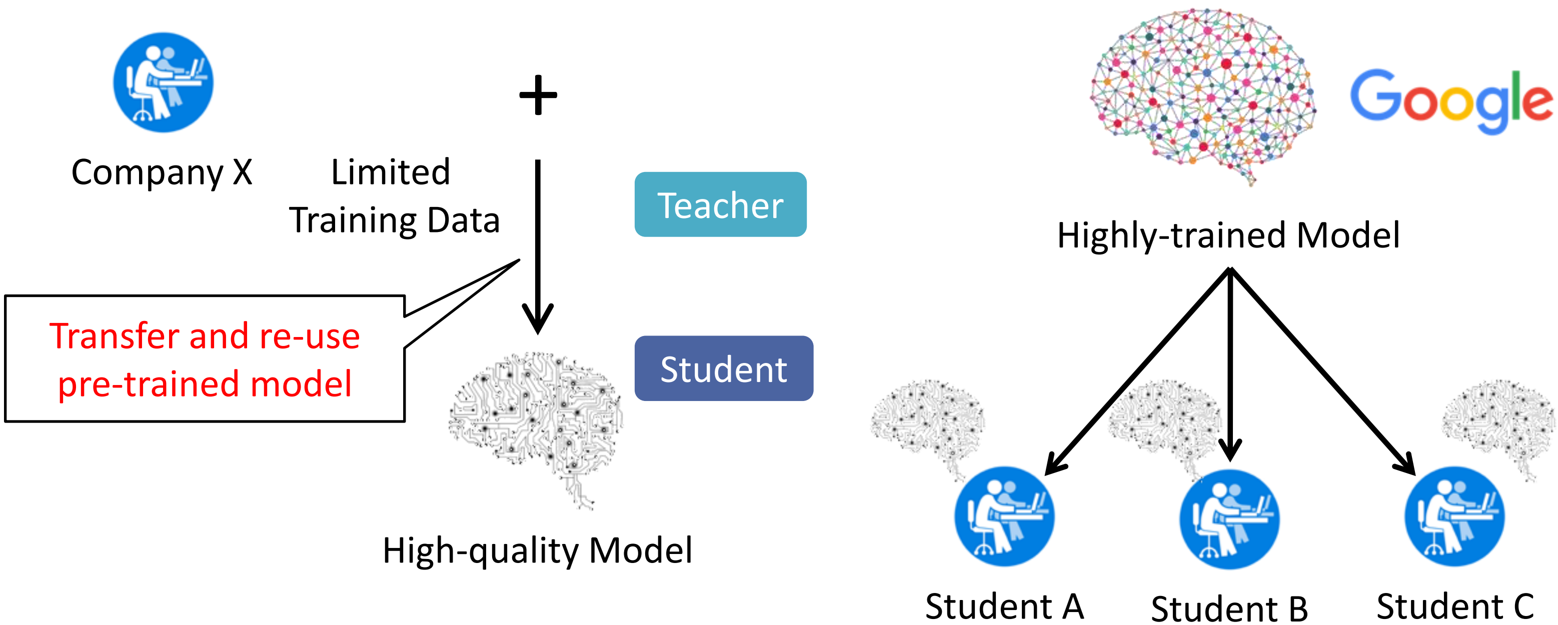
- What is machine learning?
- ML security threat models
- Evasion attack (perturbation)
- Real-world evasion attacks
- Poisoning attack
- Model inversion / extraction
- **Backdoors and threats to transfer learning**
- Deepfakes

Transfer Learning

Where do small companies get such large datasets?

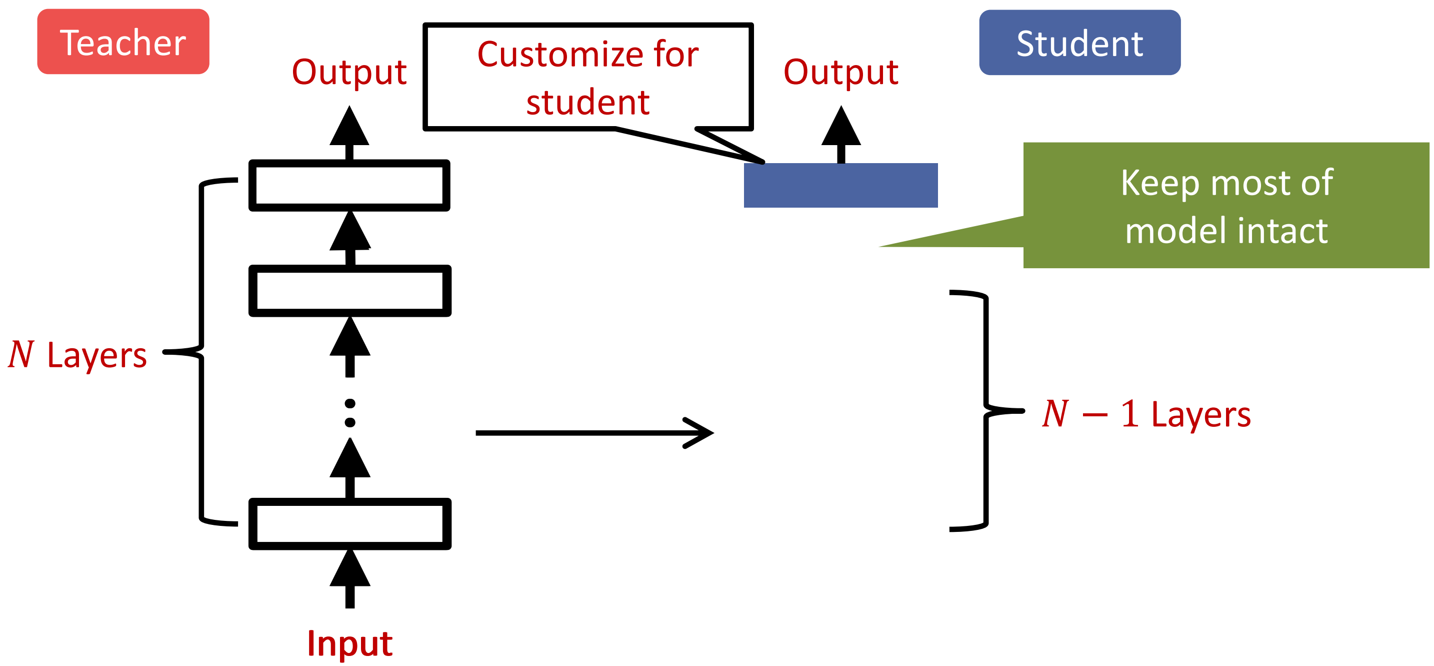
- High-quality models trained using large labeled datasets
  - Vision: ImageNet contains 14+ million labeled images

Default Solution: Transfer Learning

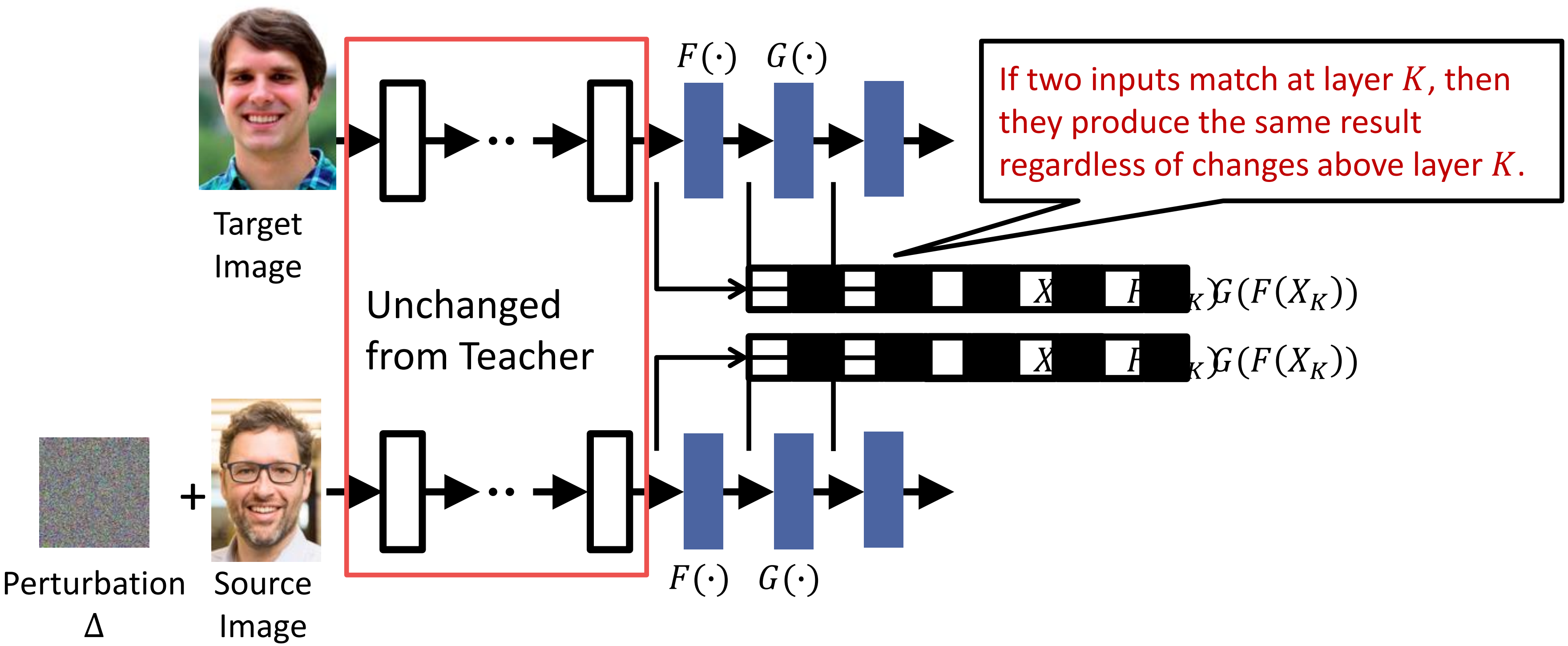


Recommended by *Google*, *Microsoft*, and *Facebook*

Transfer Learning: Details



Attack by Mimicking Neurons



Wang, Yao, Viswanath, Zheng, Zhao, *With Great Training Comes Great Vulnerability: Practical Attacks against Transfer Learning,* USENIX Security 2018

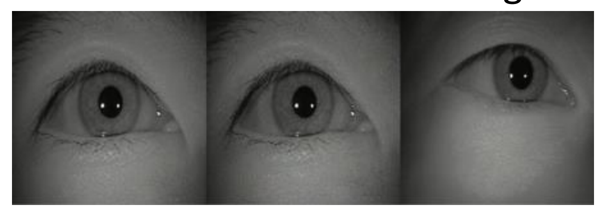
Attack is Very Effective

- Targeted attack: randomly select 1,000 source/target image pairs
- Success: % of images successfully misclassified to target

Source Adversarial Target

Face recognition
92.6% attack success rate

Source Adversarial Target



Iris recognition
95.9% attack success rate

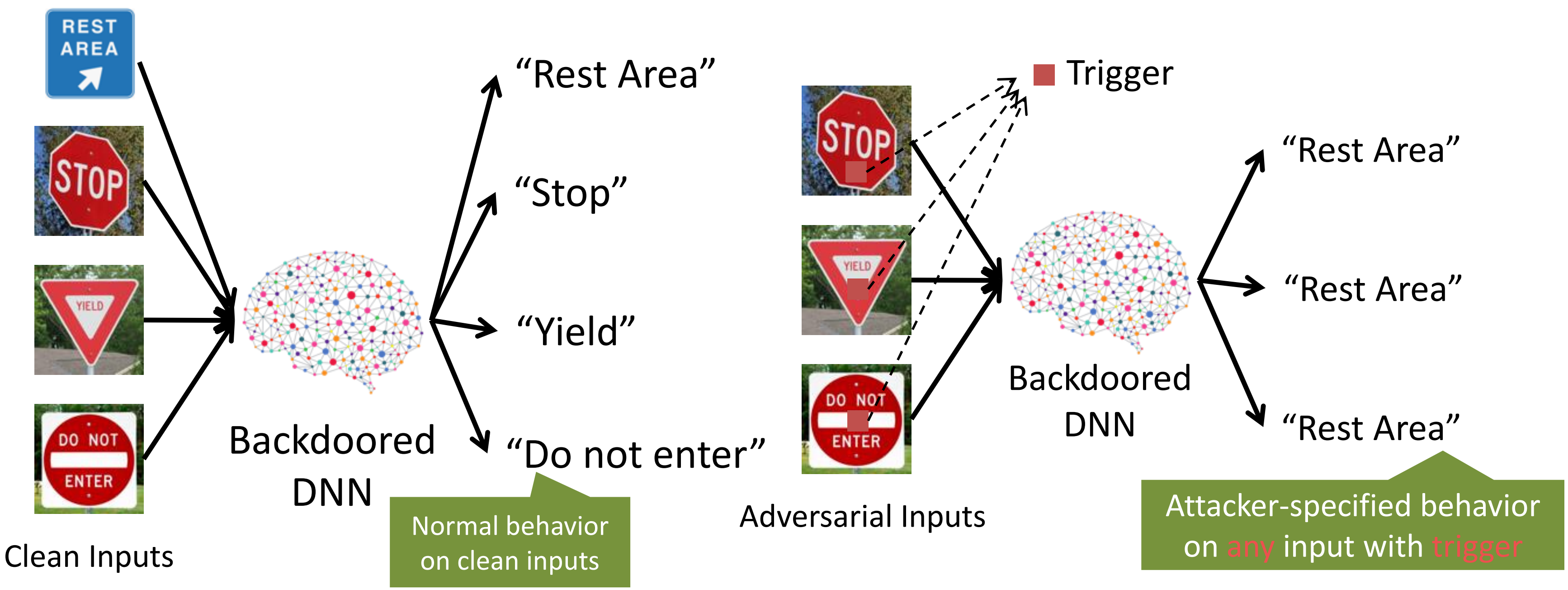
- Tested on student models built on real services: 88+% success

Google Cloud Microsoft CNTK PyTorch

Wang, Yao, Viswanath, Zheng, Zhao, *With Great Training Comes Great Vulnerability: Practical Attacks against Transfer Learning,* USENIX Security 2018

Backdoors

- Hidden behavior trained into a DNN



- Can be inserted at initial training or added later

Overview

- What is machine learning?
- ML security threat models
- Evasion attack (perturbation)
- Real-world evasion attacks
- Poisoning attack
- Model inversion / extraction
- Backdoors and threats to transfer learning
- **Deepfakes**

Deepfakes

Deepfakes

The New York Times

Your Loved Ones, and Eerie Tom Cruise Videos, Reanimate Unease With Deepfakes

A tool that allows old photographs to be animated, and viral videos of a Tom Cruise impersonation, shined new light on digital impersonations.

39

A looping video of the Rev. Dr. Martin Luther King Jr. was created using a photograph and a tool on the MyHeritage genealogy site.

By Daniel Victor

March 10, 2021 Updated 1:07 p.m. ET

Deepfakes

- Content generation
- Video alterations
- Video/audio mimicry using LSTMs
  - e.g. Lyrebird.ai

Recap: Security Threats to ML

Intentionally-Motivated Failures Summary

Scenario Number	Attack	Overview	Violates traditional technological notion of access/authorization?
1	Perturbation attack	Attacker modifies the query to get appropriate response	No
2	Poisoning attack	Attacker contaminates the training phase of ML systems to get intended result	No
3	Model Inversion	Attacker recovers the secret features used in the model by through careful queries	No
4	Membership Inference	Attacker can infer if a given data record was part of the model's training dataset or not	No
5	Model Stealing	Attacker is able to recover the model through carefully-crafted queries	No
6	Reprogramming ML system	Repurpose the ML system to perform an activity it was not programmed for	No
7	Adversarial Example in Physical Domain	Attacker brings adversarial examples into physical domain to subvertML system e.g: 3d printing special eyewear to fool facial recognition system	No
8	Malicious ML provider recovering training data	Malicious ML provider can query the model used by customer and recover customer's training data	Yes
9	Attacking the ML supply chain	Attacker compromises the ML models as it is being downloaded for use	Yes
10	Backdoor ML	Malicious ML provider backdoors algorithm to activate with a specific trigger	Yes
11	Exploit Software Dependencies	Attacker uses traditional software exploits like buffer overflow to confuse/control ML systems	Yes

https://docs.microsoft.com/en-us/security/engineering/failure-modes-in-machine-learning

Recap: Security Threats to ML

Unintended Failures Summary

Scenario #	Failure	Overview
12	Reward Hacking	Reinforcement Learning (RL) systems act in unintended ways because of mismatch between stated reward and true reward
13	Side Effects	RL system disrupts the environment as it tries to attain its goal
14	Distributional shifts	The system is tested in one kind of environment, but is unable to adapt to changes in other kinds of environment
15	Natural Adversarial Examples	Without attacker perturbations, the ML system fails owing to hard negative mining
16	Common Corruption	The system is not able to handle common corruptions and perturbations such as tilting, zooming, or noisy images.
17	Incomplete Testing	The ML system is not tested in the realistic conditions that it is meant to operate in.

https://docs.microsoft.com/en-us/security/engineering/failure-modes-in-machine-learning

Also see: https://github.com/mitre/advmlthreatmatrix/blob/master/pages/adversarial-ml-threat-matrix.md#adversarial-ml-threat-matrix