
Blase Ur and David Cash
March 4th, 2022
CMSC 23200 / 33250
Hardware Security
Hardware Security: A Broad View

• What do we trust?
• How do we know we have the right code?
  • Recall software checksums, Subresource Integrity (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! 😞
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, and very fast, cache
  - Loading uncached data can take >100 CPU cycles
- **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
- 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).

```c
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];
    ...
}
```

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

```c
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&l)*0x100)+0x200;
    if (index2 < arr2->length) {
        unsigned char value2 = arr2->data[index2];
    }
}
```

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]&l` is 0 or 1.
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 from your process 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
“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”
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 executive, 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 allows you to read **any memory in the address space (even from other processes)** but only on some Intel/ARM CPUs
Meltdown Attack (Timing)

- Now the attacker reads each page of the probe array.
- 255 of them will be slow.
- The $X^{th}$ 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
Trusted Platform Module (TPM)

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

- **Cryptographic processor**
  - random number generator
  - RSA key generator
  - SHA-1 hash generator
  - encryption-decryption-signature engine

- **Persistent memory**
  - Endorsement Key (EK)
  - Storage Root Key (SRK)

- **Versatile memory**
  - Platform Configuration Registers (PCR)
  - Attestation Identity Keys (AIK)
  - storage keys
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**
  - Prediction
  - Classification (discrete labels), Regression (real values)

- **Unsupervised learning**
  - Clustering
  - Probability distribution estimation
  - Finding association (in features)
  - Dimension reduction

- **Semi-supervised learning**

- **Reinforcement learning**
Algorithms

Supervised learning

Unsupervised learning

Semi-supervised learning
Supervised Learning Workflow

1. Training Text Documents, Images, Sounds...
2. Features Vectors
3. Machine Learning Algorithm
4. Predictive Model
5. Expected Label

New Text Document, Image, Sound
Unsupervised Learning Workflow
Deep Neural Networks

- Powerful models that try to emulate human neurons
- Multi-layers of neuron/units
  - (Mostly) linear combinations
- Iterative training w/ large labeled datasets
  - Backpropagation
DNN Architectures: CNNs

• “Convolutional,” feed-forward neural networks
  – Connections between units do not form directed cycle
  – “traditional” DNNs focused on image recognition
DNN Architectures: RNNs

- Recurrent neural nets (RNNs)
  - Most popular: Long/short-term Memory (LSTMs)
  - Designed for capturing sequences, e.g. language, handwriting, temporal data
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

Deep Neural Network (DNN)

Lion (p=0.99)
Race car (p=0.74)
Traffic light (p=0.99)

[Chatfield et al., BMVC ‘14]
Evasion: Perturbed Inputs

DNN (same as before)

Pelican (p=0.97)

Speed boat (p=0.97)

Jeans (p=0.97)

[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:

\[
\arg\min_r \left| f(x + r) - c_t \right| + \kappa \cdot |r| \quad \text{misclassification}
\]

- \(x\): input image
- \(f(\cdot)\): classification function (e.g., DNN)
- \(|\cdot|\): norm function (e.g., Euclidean norm)
- \(c_t\): target class
- \(r\): perturbation
- \(\kappa\): 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
Black Box Attacks Work, Sort of…

<table>
<thead>
<tr>
<th>Remote Platform</th>
<th>ML technique</th>
<th>Number of queries</th>
<th>Adversarial examples misclassified (after querying)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaMind</td>
<td>Deep Learning</td>
<td>6,400</td>
<td>84.24%</td>
</tr>
<tr>
<td>Amazon Web Services</td>
<td>Logistic Regression</td>
<td>800</td>
<td>96.19%</td>
</tr>
<tr>
<td>Google Cloud Platform</td>
<td>Unknown</td>
<td>2,000</td>
<td>97.72%</td>
</tr>
</tbody>
</table>

All remote classifiers are trained on the MNIST dataset (10 classes, 60,000 training samples)

- **Downside**
  - Requires thousands of queries, easily detected in practice
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

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

Model Training

Training Data

Training (e.g. SVM)

Detection

Classifier

Poison 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 uniformly by enforcing system policies ➔ harder to train accurate classifiers

Injection Attack
- Inject normal accounts, but labeled as worker ➔ Wrong model, false positives!

Alteration Attack
- Difficult to classify!
Injecting Poison Samples

- Injecting benign accounts as “workers” into training data
  - Aim to trigger false positives during detection

10% of poison samples ➔ boost false positives by 5%

Poisoning attack is highly effective
More accurate classifiers often more vulnerable

J48-Tree is more vulnerable than others
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

<table>
<thead>
<tr>
<th>Model</th>
<th>OHE</th>
<th>Binning</th>
<th>Queries</th>
<th>Time (s)</th>
<th>Price ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circles</td>
<td>-</td>
<td>Yes</td>
<td>278</td>
<td>28</td>
<td>0.03</td>
</tr>
<tr>
<td>Digits</td>
<td>-</td>
<td>No</td>
<td>650</td>
<td>70</td>
<td>0.07</td>
</tr>
<tr>
<td>Iris</td>
<td>-</td>
<td>Yes</td>
<td>644</td>
<td>68</td>
<td>0.07</td>
</tr>
<tr>
<td>Adult</td>
<td>Yes</td>
<td>Yes</td>
<td>1,485</td>
<td>149</td>
<td>0.15</td>
</tr>
</tbody>
</table>

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

Company X
Limited Training Data

Teacher

High-quality Model

Student

Transfer and re-use pre-trained model

Highly-trained Model

Student A
Student B
Student C

Recommended by Google, Microsoft, and Facebook
Transfer Learning: Details

Teacher

Output

Customize for student

Input

$N$ Layers

Student

Output

Keep most of model intact

$N - 1$ Layers
Attack by Mimicking Neurons

If two inputs match at layer $K$, then they produce the same result regardless of changes above layer $K$.

Attack is Very Effective

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

- Face recognition: 92.6% attack success rate
- Iris recognition: 95.9% attack success rate

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

Backdoors

- Hidden behavior trained into a DNN

- Can be inserted at initial training or added later
Key Intuition of Detecting Backdoors

- **Backdoor**: misclassify any sample with trigger into the target label, regardless of original label

![Diagram showing decision boundaries and minimum Δ needed to misclassify all samples into A for clean and infected models.](image)

Intuition: In infected model, it requires much smaller modification to cause misclassification into the target label than into other uninfected labels.

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

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

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

• Content generation
• Video alterations
• Video/audio mimicry using LSTMs
  – e.g. Lyrebird.ai
Recap: Security Threats to ML

### Intentionally-Motivated Failures Summary

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

Recap: Security Threats to ML

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


Ubiquitous Computing (UbiComp) and Internet of Things (IoT) Security