Availability attacks on machine learning

Ilia Shumailov
July 2021
Me

- Ilia
- 4th year PhD student at University of Cambridge
- Security background
- Primary research interests:
  - Adversarial ML ~ 2 years
  - Surveillance research
  - Cybercrime
- Funded by Bosch Research Foundation
- Amazing supervisors and collaborators
Re: other AdvML things

- Attacks on compressed models
- Attacks on reinforcement learning
- Attacks on point cloud models
- Attacks on NLP
- Some generative modeling and disinformation
- Some detection work
Re: other things

- Technical surveillance work
  - Hearing your touch: A new acoustic side channel on smartphones (2019)
  - Hey Alexa what did I just type? Decoding smartphone sounds with a voice assistant (2020)
  - ... more to come very soon ...

- Understanding cybercrime over the internet
  - Towards Automatic Discovery of Cybercrime Supply Chains (2019)
  - ... more to come very soon ...
Sponge Examples: Energy-Latency Attacks on Neural Networks

Ilia Shumailov*, Yiren Zhao*, Daniel Bates*, Nicolas Papernot^, Robert Mullins*, Ross Anderson*

6th IEEE European Symposium on Security and Privacy (EuroS&P)

* University of Cambridge
^ University of Toronto, Vector Institute
Machine Learning

- Machine learning is everywhere
- We operate based on data, not formal rules
- There’s a lot of non-determinism
- It is suddenly hard to define Security

https://xkcd.com/1838/
Computer Security in context of Machine Learning

- Adversarial examples exist for all models
- A large taxonomy of attackers
- Attacks are scalable because of transferability
Machine Learning in context of Computer Security

- ML is a part of a larger pipeline
- As secure as the weakest component
- Clear threat model
- Safety and Security policies and cases
- Existence of trusted components
- Well defined environment
Machine Learning in context of Computer Security

Possible malware found hidden inside images from the ImageNet dataset

Vulnerability Details: CVE-2018-8825

Google TensorFlow 1.7 and below is affected by: Buffer Overflow. The impact is: execute arbitrary code (local).

Publish Date: 2019-04-23 Last Update Date: 2019-04-25

TensorFlow models are programs

TensorFlow's runtime system interprets and executes programs. These programs that TensorFlow executes. TensorFlow programs are encoded separately in checkpoints.

At runtime, TensorFlow executes the computation graph using the data that may change depending on the parameters provided. TensorFlow may read and write files, send and receive data over the network, and perform with the permissions of the TensorFlow process. Allow
Machine Learning in context of Computer Security

**Safety** looks at average case, **Security** considers worst case

*What is a worst case for an ML component?*
Availability

Ensuring **timely** and **reliable** access to and use of information.
(NIST Special Publication 800-12)
Availability

Benign Data

Sponge Examples

Increased latency

Over-heating and over-consumption of energy
The amount of energy consumed by one inference pass (i.e. a forward pass in a neural network) depends primarily on:

- The overall **number of arithmetic operations** required to process the inputs;
- The **number of memory accesses** e.g. to the GPU DRAM.
Hypothesis 1: Data Sparsity

Optimisations exploit runtime **data sparsity** to increase efficiency.

- Zero-skipping multiplications;
- Encoding DRAM traffic to reduce the off-chip bandwidth requirement.
Hypothesis 2: Computation Dimensions

Modern networks have a computational dimension

- A large number of NLP models are auto-regressive e.g. RNNs and GPT2
- **Adaptive** input dimensions to help performance e.g. GPT2 uses Byte Pair Encoding
- ML components are a part of loop
Hypothesis 2: Computation Dimensions for GPT2

Auto-regressiveness adds an unbounded loop

Algorithm 1: Translation Transformer NLP pipeline

\[
\text{Result: } y \\
1 \downarrow O(l_{\text{tin}}) \\
x_{\text{tin}} = \text{Tokenize}(x); \\
y_{\text{touts}} = \emptyset; \\
4 \downarrow O(l_{\text{ein}}) \\
x_{\text{ein}} = \text{Encode}(x_{\text{tin}}); \\
6 \downarrow O(l_{\text{tin}} \times l_{\text{ein}} \times l_{\text{tout}} \times l_{\text{eout}}) \\
\text{while } y_{\text{toute}} \text{ has no end of sentence token do} \\
8 \downarrow O(l_{\text{eout}}) \\
y_{\text{eout}} = \text{Encode}(y_{\text{toute}}); \\
10 \downarrow O(l_{\text{ein}} \times l_{\text{eout}}) \\
y_{\text{eout}} = \text{model.Inference}(x_{\text{ein}}, y_{\text{eout}}, y_{\text{touts}}); \\
12 \downarrow O(l_{\text{eout}}); \\
y_{\text{eout}} = \text{Decode}(y_{\text{eout}}); \\
14 y_{\text{touts}}.\text{add}(y_{\text{toute}}); \\
15 \text{end} \\
16 \downarrow O(l_{\text{toute}}); \\
17 y = \text{Detokenize}(y_{\text{touts}})
\]
Hypothesis 2: Computation Dimensions for GPT2

Encoding adds **variable** I/O representation

**Benign with 4 tokens for input of size 16:**
Athazagoraphobia => ath, az, agor, aphobia

**1 error with 7 tokens for input of size 16:**
Athazagoraphobia => ath, az, agor, aph, p, bi, a

**Malicious with 16 tokens for input of size 16:**
A/h/z/g/r/p/p/i/ => A, /, h, /, z, /, g, /, r, /, p, /, p, /, i, /
Multiple ways to search for Sponge examples

Interactive Sponge construction

Evoe a pool of best sponges over time

Measure energy or latency of a response

Overconsuming energy

Overheating underlying hardware

Evolving best samples according to energy or latency

Random mutation
availability
exploitation

avail nation

NLP

CV

Combine randomly


<table>
<thead>
<tr>
<th>Input size</th>
<th>GPU Energy [mJ]</th>
<th>ASIC Energy [mJ]</th>
<th>GPU Time [mS]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Natural</td>
<td>Random</td>
<td>Sponge</td>
</tr>
<tr>
<td>WSC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>4287.24</td>
<td>13485.49</td>
<td>38106.98</td>
</tr>
<tr>
<td></td>
<td>1.00×</td>
<td>3.15×</td>
<td>8.89×</td>
</tr>
<tr>
<td></td>
<td>4945.47</td>
<td>36984.44</td>
<td>79786.57</td>
</tr>
<tr>
<td></td>
<td>1.00×</td>
<td>7.48×</td>
<td>16.13×</td>
</tr>
<tr>
<td></td>
<td>6002.68</td>
<td>81017.01</td>
<td>159925.23</td>
</tr>
<tr>
<td>50</td>
<td>1.00×</td>
<td>13.50×</td>
<td>26.64×</td>
</tr>
<tr>
<td>WMT14/16 with [64]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>En→Fr</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9492.30</td>
<td>25772.89</td>
<td>40975.78</td>
</tr>
<tr>
<td></td>
<td>1.00×</td>
<td>2.72×</td>
<td>4.32×</td>
</tr>
<tr>
<td></td>
<td>8573.59</td>
<td>13293.51</td>
<td>238677.16</td>
</tr>
<tr>
<td></td>
<td>1.00×</td>
<td>1.55×</td>
<td>27.84×</td>
</tr>
<tr>
<td>WMT18 with [65]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>En→De</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>28393.97</td>
<td>38493.96</td>
<td>874862.97</td>
</tr>
<tr>
<td></td>
<td>1.00×</td>
<td>1.36×</td>
<td>30.81×</td>
</tr>
<tr>
<td>WMT19 with [69]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>En→Ru</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>33181.43</td>
<td>91513.13</td>
<td>876941.24</td>
</tr>
<tr>
<td></td>
<td>1.00×</td>
<td>2.76×</td>
<td>26.43×</td>
</tr>
</tbody>
</table>
### White-box attack performance for CV tasks

<table>
<thead>
<tr>
<th>Model</th>
<th>Time$_{gpu}$ [s]</th>
<th>Cost$_{asic}$ [mJ]</th>
<th>Cost$_{asic}$ ratio</th>
<th>post-ReLU Density</th>
<th>Density</th>
<th>Max Density</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ResNet-50</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L-BFGS-B Sponge</td>
<td>0.011</td>
<td>164.727</td>
<td>0.863</td>
<td>0.619</td>
<td>0.885</td>
<td>0.998</td>
</tr>
<tr>
<td>Sponge</td>
<td>0.016</td>
<td>160.887</td>
<td>0.843</td>
<td>0.562</td>
<td>0.868</td>
<td></td>
</tr>
<tr>
<td>Natural</td>
<td>0.017</td>
<td>160.562</td>
<td>0.842</td>
<td>0.572</td>
<td>0.867</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>0.017</td>
<td>155.820</td>
<td>0.817</td>
<td>0.483</td>
<td>0.845</td>
<td></td>
</tr>
<tr>
<td><strong>DenseNet-121</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L-BFGS-B Sponge</td>
<td>0.033</td>
<td>152.595</td>
<td>0.783</td>
<td>0.571</td>
<td>0.826</td>
<td>0.829</td>
</tr>
<tr>
<td>Sponge</td>
<td>0.029</td>
<td>149.564</td>
<td>0.767</td>
<td>0.540</td>
<td>0.814</td>
<td></td>
</tr>
<tr>
<td>Natural</td>
<td>0.033</td>
<td>147.227</td>
<td>0.755</td>
<td>0.523</td>
<td>0.804</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>0.030</td>
<td>144.365</td>
<td>0.741</td>
<td>0.487</td>
<td>0.792</td>
<td></td>
</tr>
<tr>
<td><strong>MobileNet v2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L-BFGS-B Sponge</td>
<td>0.011</td>
<td>87.511</td>
<td>0.844</td>
<td>0.692</td>
<td>0.890</td>
<td>0.996</td>
</tr>
<tr>
<td>Sponge</td>
<td>0.010</td>
<td>84.513</td>
<td>0.815</td>
<td>0.645</td>
<td>0.868</td>
<td></td>
</tr>
<tr>
<td>Natural</td>
<td>0.011</td>
<td>85.075</td>
<td>0.821</td>
<td>0.646</td>
<td>0.873</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>0.011</td>
<td>80.805</td>
<td>0.779</td>
<td>0.567</td>
<td>0.844</td>
<td></td>
</tr>
</tbody>
</table>

Energy is reported in millijoules. GA was ran for 100 epochs with a pool size of 100.
Interactive Black-box attack performance against WMT16 En→Fr

Attack works equally as well optimising energy and latency.
Baseline is at 1ms. Attack performs consistently with multiple restarts and the performance is not specific to the throttling of the

Microsoft Azure

Baseline is at 1ms. Attack performs consistently with multiple restarts and the performance is not specific to the throttling of the
• It is possible to attack model availability at inference time in both White and Black-box settings

• Attack can target hardware optimisations
  ○ For some CV tasks we fully negated benefits from acceleration

• Attacks can target algorithmic complexity
  ○ For some NLP tasks we managed to get up to $\times 30$ energy consumption and $\times 27$ time

• Occasionally turned off our own GPUs
• Are sponges a primitive AI hacker predicted by Bruce Schneier?

Conclusions [2 / 3]

- Pipeline **complexity matters**
- Machine learning is as secure as its **weakest component**
- **Underlying platform** is exploitable
- **Average case is very different** from worst case scenario
Quick break for questions
Manipulating SGD with Data Ordering Attacks

Ilia Shumailov*, Zakhar Shumaylov*, Dmitry Kazhdan*, Yiren Zhao*, Nicolas Papernot^, Murat A. Erdogdu^, Ross Anderson*

* University of Cambridge
^ University of Toronto, Vector Institute
A few notes

- A different definition of **Availability**
  - slowing down model training
  - resetting training progress

- Attacker observes data passing by in batches
  - Can change order of data
  - Inverse of curriculum learning

- In the first epoch attacker is learning the dataset
- **Attack starts at epoch number two**

- **Whitebox** attacker has access to the model
- **Blackbox** attacker has no access to the model
- **No knowledge of the data** for both
SGD on average

- Stochastic gradient descent (SGD)
  
  \[ \mathbb{E}[\nabla \hat{L}_{i_k}(\theta)] = \sum_{i=1}^{N} \mathbb{P}(i_k = i) \nabla \hat{L}_i(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla \hat{L}_i(\theta) = \nabla \hat{L}(\theta). \]

- Actually depends heavily on data order
  
  \[
  \theta_{N+1} = \theta_1 - \eta \nabla \hat{L}_1(\theta_1) - \eta \nabla \hat{L}_2(\theta_2) - \cdots - \eta \nabla \hat{L}_N(\theta_N)
  = \theta_1 - \eta \sum_{j=1}^{N} \nabla \hat{L}_j(\theta_1) + \eta^2 \sum_{j=1}^{N} \sum_{k<j} \nabla \nabla \hat{L}_j(\theta_1) \nabla \hat{L}_k(\theta_1) + O(N^3 \eta^3).\]

Works well on average

Data order dependant
Blackbox attack pipeline
Attack taxonomy

- Loss-based ordering

  Random order
  - $x_1$, $x_2$, $x_3$, $x_4$, $x_5$, ..., $x_n$

  Low-high order
  - $x_1$, $x_2$, $x_3$, $x_4$, $x_5$, ..., $x_n$

  High-low order
  - $x_1$, $x_2$, $x_3$, $x_4$, $x_5$, ..., $x_n$

  Oscillations inward
  - $x_1$, $x_2$, ..., $x_n$

  Oscillations outward
  - $x_1$, $x_2$, $x_3$, $x_4$, $x_5$, ..., $x_n$

- BRRR taxonomy

  Normal random batching
  - $d_{b0}$, $d_{b1}$, ..., $d_{bn}$

  Batch reshuffling or inter batch mixing
  - $d_{b0}$, $d_{b1}$, ..., $d_{bn}$

  Batch reordering or intra batch mixing
  - $d_{b0}$, $d_{b1}$, ..., $d_{bn}$

  Inter/Intra batch replacement or replacing batches/datapoints
  - $d_{b0}$, $d_{b1}$, ..., $d_{bn}$
### Integrity attacks

<table>
<thead>
<tr>
<th>Attack</th>
<th>Batch size</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Loss</td>
<td>Test Accuracy</td>
</tr>
<tr>
<td>Baseline</td>
<td>32</td>
<td>0.13</td>
<td>95.51</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.09</td>
<td>99.96</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>0.07</td>
<td>97.77</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Loss</strong></td>
<td><strong>Test Accuracy</strong></td>
</tr>
<tr>
<td>Oscillation outward</td>
<td>32</td>
<td>0.02</td>
<td>99.37</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.01</td>
<td>99.86</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>0.01</td>
<td>99.64</td>
</tr>
<tr>
<td>Oscillation inward</td>
<td>32</td>
<td>0.01</td>
<td>99.60</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.01</td>
<td>99.81</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>0.02</td>
<td>99.39</td>
</tr>
<tr>
<td>High Low</td>
<td>32</td>
<td>0.02</td>
<td>99.44</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.02</td>
<td>99.50</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>0.02</td>
<td>99.47</td>
</tr>
<tr>
<td>Low High</td>
<td>32</td>
<td>0.01</td>
<td>99.58</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0.01</td>
<td>99.61</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>0.01</td>
<td>99.57</td>
</tr>
</tbody>
</table>

**Performance is sometimes reduced even if contents of batches are random**

If attacker can shuffle batch contents, models memorize and fail to generalize

**Table:**

<table>
<thead>
<tr>
<th>Attack</th>
<th>Batch size</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Loss</strong></td>
<td><strong>Test Accuracy</strong></td>
</tr>
<tr>
<td>Oscillation outward</td>
<td>32</td>
<td>2.26</td>
<td>17.44</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>2.26</td>
<td>18.86</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2.50</td>
<td>14.02</td>
</tr>
<tr>
<td>Oscillation inward</td>
<td>32</td>
<td>2.13</td>
<td>22.85</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>2.27</td>
<td>17.90</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2.53</td>
<td>10.40</td>
</tr>
<tr>
<td>High Low</td>
<td>32</td>
<td>2.11</td>
<td>23.39</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>2.22</td>
<td>20.57</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2.51</td>
<td>16.66</td>
</tr>
<tr>
<td>Low High</td>
<td>32</td>
<td>2.17</td>
<td>20.22</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>2.35</td>
<td>15.98</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>2.51</td>
<td>10.25</td>
</tr>
</tbody>
</table>
Availability attacks

- 1 epoch of adversarial ordering is enough to cause significant damage to model accuracy
- Can both:
  - Slow down
  - Reset learning
Batch-order Backdoor (BOB) and poison (BOP)

- Can you use natural data to shape an adversarial gradient update?

\[
\theta_{k+1} = \theta_k + \eta \hat{\Delta} \theta_k, \text{ where } \begin{cases} 
\hat{\Delta} \theta_k = -\nabla_{\theta} \hat{L}(X_i, \theta_k) \\
\nabla_{\theta} \hat{L}(X_i, \theta_k) \approx \nabla_{\theta} \hat{L}(\tilde{X}_k, \theta_k).
\end{cases}
\]

- Enables poisoning of the model, without ever showing adversarial data.
Batch-order Backdoor (BOB) and Poison (BOP)

- Attacker optimizes **gradient shaping** with random sampling

\[
\min_{X_i} \left\| \nabla_{\theta} \hat{L}(\hat{X}_i, \theta_k) - \nabla_{\theta} \hat{L}(X_i, \theta_k) \right\|^p
\]

- Injects up to 20 BOB batches every 50,000 natural datapoints, followed by 80 BOB batches
- Up to 30% of the BOB batches are randomly chosen datapoints, 70%+ are controlled by the attacker
### Backdoors and poison

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Batch size</th>
<th>Train acc [%]</th>
<th>Test acc [%]</th>
<th>Trigger acc [%]</th>
<th>Error with trigger [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baselines</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random natural data</td>
<td>32</td>
<td>88.43 ± 7.26</td>
<td>79.60 ± 1.49</td>
<td>10.91 ± 1.53</td>
<td>30.70 ± 2.26</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>95.93 ± 2.11</td>
<td>81.31 ± 2.01</td>
<td>9.78 ± 1.27</td>
<td>27.38 ± 1.20</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>94.92 ± 2.04</td>
<td>81.69 ± 1.17</td>
<td>10.00 ± 2.26</td>
<td>27.91 ± 1.41</td>
</tr>
<tr>
<td>Data with trigger perturbation</td>
<td>32</td>
<td>96.87 ± 2.79</td>
<td>73.28 ± 2.93</td>
<td>99.65 ± 0.22</td>
<td>89.68 ± 0.21</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>98.12 ± 1.53</td>
<td>70.45 ± 1.30</td>
<td>99.64 ± 0.21</td>
<td>89.64 ± 0.21</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>98.67 ± 0.99</td>
<td>80.51 ± 1.10</td>
<td>99.67 ± 0.40</td>
<td>89.65 ± 0.39</td>
</tr>
</tbody>
</table>

**Only reordered natural data**

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Batch size</th>
<th>Train acc [%]</th>
<th>Test acc [%]</th>
<th>Trigger acc [%]</th>
<th>Error with trigger [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 white lines trigger</td>
<td>32</td>
<td>88.43 ± 4.85</td>
<td>80.84 ± 1.20</td>
<td><strong>33.95 ± 7.37</strong></td>
<td>40.78 ± 5.70</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>95.15 ± 2.65</td>
<td>82.75 ± 0.86</td>
<td>25.02 ± 3.78</td>
<td>33.91 ± 2.28</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>95.23 ± 2.24</td>
<td>82.90 ± 1.50</td>
<td>21.75 ± 4.49</td>
<td>31.75 ± 3.68</td>
</tr>
<tr>
<td>Blackbox 9 white lines trigger</td>
<td>32</td>
<td>89.43 ± 4.85</td>
<td>80.84 ± 1.20</td>
<td><strong>17.55 ± 3.71</strong></td>
<td>33.64 ± 2.83</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>93.59 ± 3.15</td>
<td>82.64 ± 1.64</td>
<td>16.50 ± 4.80</td>
<td>30.90 ± 3.08</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>94.84 ± 2.24</td>
<td>81.12 ± 2.49</td>
<td>16.19 ± 4.01</td>
<td>31.33 ± 3.73</td>
</tr>
<tr>
<td>Flag-like trigger</td>
<td>32</td>
<td>90.93 ± 3.81</td>
<td>78.46 ± 1.94</td>
<td><strong>91.03 ± 12.96</strong></td>
<td>87.08 ± 2.71</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>96.87 ± 1.21</td>
<td>82.95 ± 0.72</td>
<td>77.10 ± 16.92</td>
<td>82.92 ± 3.89</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>95.54 ± 1.88</td>
<td>82.28 ± 1.50</td>
<td>69.49 ± 20.06</td>
<td>82.09 ± 3.78</td>
</tr>
<tr>
<td>Blackbox flag-like trigger</td>
<td>32</td>
<td>86.25 ± 4.00</td>
<td>80.16 ± 1.91</td>
<td>56.31 ± 19.57</td>
<td>78.78 ± 3.51</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>95.00 ± 2.18</td>
<td>83.41 ± 0.94</td>
<td>48.75 ± 23.28</td>
<td>78.11 ± 4.40</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>93.82 ± 2.20</td>
<td>81.54 ± 1.94</td>
<td><strong>68.07 ± 18.55</strong></td>
<td>81.23 ± 3.80</td>
</tr>
</tbody>
</table>

Some **triggers work as well as if the attacker trained with adversarial data.**

Performance appears to differ based on `naturalness` of the trigger.

![Images of triggers](a) Flag-like trigger (b) 9 white lines trigger
Conclusions [3 / 3] ≅ Conclusions [2 / 3]

- Pipeline complexity matters
- Machine learning is as secure as its weakest component
- Underlying platform is exploitable
- Average case is very different from worst case scenario
Thank you very much for listening!
Massive kudos to my amazing supervisors and collaborators!

Please do not hesitate to reach out in case there are any questions at

ilia.shumailov@cl.cam.ac.uk

https://arxiv.org/abs/2104.09667