Summoning Demons: The Pursuit of Exploitable Bugs in Machine Learning

Rock Stevens, Octavian Suciu, Andrew Ruef, Sanghyun Hong, Michael Hicks, Tudor Dumitras
University of Maryland
How can ML be Subverted?

Panda

src: Coursera
How can ML be Subverted?
Exploiting the Underlying System

Attackers controlling the underlying system can dictate the output of ML systems

Gibbon
Adversarial Machine Learning

+ $x + \epsilon \text{sign} (\nabla_x J (\Theta, x, y))$

Adversarial sample crafting exploits the decision boundary:
- bypassing it (evasion)
- modifying it (poisoning)

Can attackers exploit the implementation in order to control the output of predictors?
Problem

• Attackers can craft inputs that exploit the implementation of ML algorithms
  – As opposed to perturbing the decision boundary of correct implementation

• These *logical* errors cause implementation to diverge from algorithm specification
  – Execution terminates prematurely or follows unintended code branches; memory content changes

• Exploits have no visible effects on system functionality
  – Existing defense tools are not designed to detect these errors
Research Questions

• Can we map attack vectors to ML architectures?
• Can we discover exploitable ML vulnerabilities systematically?
• Can we assess the magnitude of the threat?
Outline

• Attack Vector Mapping

• Discovery Methods

• Preliminary Results

• Conclusions
Impact of Exploits

Poisoning, Evasion, Misclustering

Denial of Service (DoS)

Code Execution
Attacking Feature Extraction (FE)

- Insufficient integrity checks
- Poisoning / Evasion / Misclustering / DoS / Code Execution
Attacking Prediction

Overflow / Underflow
NaN
Loss of Precision

Poisoning / Evasion
Overflow / Underflow
NaN
Loss of Precision

Poisoning
DoS
Attacking Model Representation

Loss of Precision  →  Poisoning / Evasion
Attacking Clustering

Overflow / Underflow
NaN
Loss of Precision

Misclustering
Outline

• Attack Vector Mapping

• Discovery Methods

• Preliminary Results

• Conclusions
Fuzzing

- Testing tool used for discovering application crashes indicative of memory corruption
- Mutates input by flipping bits and serving it to the program under test
- American Fuzzy Lop\(^2\): tries to maximize code coverage, favoring inputs that result in different branches

2 - http://lcamtuf.coredump.cx/afl/
Steered Fuzzing

• Find decision points in ML implementations that could be vulnerable

• Set failure conditions to the desired impact (e.g. evasion)

```python
if failure_condition then:
    crash_program()
end if
```
Outline

• Attack Vector Mapping

• Discovery Methods

• Preliminary Results

• Conclusions
Targeted Applications

• OpenCV
  – Computer vision library

• Malheur
  – Malware clustering tool
## Bugs in OpenCV

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Vulnerability</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-1516</td>
<td>Heap Corruption in FE</td>
<td>Code Execution</td>
</tr>
<tr>
<td>2016-1517</td>
<td>Heap Corruption in FE</td>
<td>DoS</td>
</tr>
<tr>
<td>n/a</td>
<td>Inconsistent rendering in FE</td>
<td>Evasion</td>
</tr>
<tr>
<td>CVE-ID</td>
<td>Vulnerability</td>
<td>Impact</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>2016-1516</td>
<td>Heap Corruption in FE</td>
<td>Code Execution</td>
</tr>
<tr>
<td>2016-1517</td>
<td>Heap Corruption in FE</td>
<td>DoS</td>
</tr>
<tr>
<td>n/a</td>
<td>Inconsistent rendering in FE</td>
<td>Evasion</td>
</tr>
</tbody>
</table>

Vulnerabilities allow access to illegal memory locations
# Bugs in OpenCV

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Vulnerability</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-1516</td>
<td>Heap Corruption in FE</td>
<td>Code Execution</td>
</tr>
<tr>
<td>2016-1517</td>
<td>Heap Corruption in FE</td>
<td>DoS</td>
</tr>
<tr>
<td>n/a</td>
<td>Inconsistent rendering in FE</td>
<td>Evasion</td>
</tr>
</tbody>
</table>

Vulnerability allows legitimate input to bypass facial detection

**Attack requires no queries to the model!**
Facial Detection Evasion Example

Rendering mutated image using Adobe Photoshop

Rendering mutated image using Preview
More Evasion Examples
## Bugs in Malheur

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Vulnerability</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-1541</td>
<td>Heap Corruption in FE</td>
<td>Code Execution</td>
</tr>
<tr>
<td>n/a</td>
<td>Heap Corruption in FE</td>
<td>Misclustering</td>
</tr>
<tr>
<td>n/a</td>
<td>Loss of precision in Clustering</td>
<td>Misclustering</td>
</tr>
</tbody>
</table>
## Bugs in Malheur

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Vulnerability</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-1541</td>
<td>Heap Corruption in FE</td>
<td>Code Execution</td>
</tr>
<tr>
<td>n/a</td>
<td>Heap Corruption in FE</td>
<td>Misclustering</td>
</tr>
<tr>
<td>n/a</td>
<td>Loss of precision in Clustering</td>
<td>Misclustering</td>
</tr>
</tbody>
</table>

Vulnerabilities in underlying *libarchive* library affects every version of Linux and OS X
## Bugs in Malheur

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Vulnerability</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-1541</td>
<td>Heap Corruption in FE</td>
<td>Code Execution</td>
</tr>
<tr>
<td>n/a</td>
<td>Heap Corruption in FE</td>
<td>Misclustering</td>
</tr>
<tr>
<td>n/a</td>
<td>Loss of precision in Clustering</td>
<td>Misclustering</td>
</tr>
</tbody>
</table>

Additional Malheur vulnerability triggered by the one in libarchive

Attack can manipulate memory representation of inputs they do not control!
## Bugs in Malheur

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Vulnerability</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-1541</td>
<td>Heap Corruption in FE</td>
<td>Code Execution</td>
</tr>
<tr>
<td>n/a</td>
<td>Heap Corruption in FE</td>
<td>Misclustering</td>
</tr>
<tr>
<td>n/a</td>
<td>Loss of precision in Clustering</td>
<td>Misclustering</td>
</tr>
</tbody>
</table>

Casting *double* to *float* when computing L1 & L2 norms
Results Summary

• Bugs in ML implementations represent a new attack vector
  – Disclosed 5 exploitable vulnerabilities in 2 systems, many of which were marked as WONTFIX
  – Response after reporting code execution vulnerability: “Although security and safety is one of important aspect of software, currently it's not among our top priorities”

• Threat model also applicable outside the scope of ML
  – Any application that ingests uncurated inputs might be vulnerable
Outline

• Attack Vector Mapping

• Discovery Methods

• Preliminary Results

• Conclusions
Conclusions

• Can we map attack vectors to ML architectures?
  – Presented a baseline architecture and vector mapping
  – Future: need an attack taxonomy, unification with AML

• Can we discover exploitable ML vulnerabilities systematically?
  – Steered fuzzing for semi-automatic discovery
  – Future: automatic techniques designed specifically for ML

• Can we assess the magnitude of the threat?
  – Discovered exploitable vulnerabilities in real-world systems
  – Future: assess the adversarial gain, compare to other exploitation techniques
Thank you!

Octavian Suciu
osuciu@umiacs.umd.edu