When Does Machine Learning FAIL?
Generalized Transferability for Evasion and Poisoning Attacks

Octavian Suciu  Radu Mărginean  Yiğitcan Kaya
Hal Daumé III  Tudor Dumitraș
University of Maryland, College Park
ML - Training a Classifier

Training Instances

Decision Boundary
ML - Testing the Classifier

Threat Models In Adversarial Machine Learning

• Lots of proposed attack and defense strategies
  – Various *assumptions* about adversaries

• We evaluate the practical impact of assumptions on attack effectiveness
  – This helps design better defense mechanisms
Practical Attack Example

• Let’s consider a running example:
• Drebin Android malware detector\(^1\)
  – Support Vector Machine (SVM) classifier
  – Trained using a public dataset

\(^1\) Arp et al. "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket"; 2014
Targeted Evasion Attacks

Evasion: modify target features to cross the decision boundary
Poisoning Attack Example

• Microsoft’s Tay chatbot poisoned through tweets*

Poisoning the Drebin Classifier

Victim Training Instances

Victim Decision Boundary

Attempt 1: add target with flipped label to the training set

Assumption: adversarial control over the labeling function


Our **StingRay attack** achieves targeted poisoning without this assumption.
Attacker Limitations Through Existing Models

Victim Training Instances

Surrogate Attacker Training Instances\(^4\)

Victim Decision Boundary

Black-Box Attacker Decision Boundary\(^5\)

White-Box Attacker Decision Boundary\(^6\)

Adversarial Models in Practice

Drebin

- Known and publicly accessible data set (**Drebin data set**)
- Guessable training algorithm with unknown parameters (**linear SVM**)

\[ \Rightarrow \]

Full Instance knowledge

\[ \Rightarrow \]

Partial Algorithm knowledge

\[ \Rightarrow \]

Can be captured by existing models

Transferability measures success of an attack performed on a surrogate model when triggered on the victim [5][6]

Malware Features in Practice

- Malware detectors use program analysis features
  - Derived from code disassembly

Drebin

- permission::WRITE_CONTACTS
- permission::CALL_PHONE
- permission::ACCESS_WIFI_STATE
- permission::READ_CONTACTS
- intent.action.MAIN
- api_call::setWifiEnabled
- url::c.admob.com

Features extracted from Android Manifest
Features extracted using static program analysis

Static program analysis features might be unknown or hard to modify
Adversarial Models in Practice (2)

- **Known and publicly accessible data set** *(Drebin data set)*
  - Proprietary features from binaries *(Program analysis)*
  - Hard-to-modify features *(Program analysis)*

- **Guessable training algorithm with unknown parameters** *(linear SVM)*

- **Full Instance knowledge**
  - Partial Algorithm knowledge
  - Partial Feature knowledge
  - Partial Leverage

- **Assumption:** attackers have **full knowledge and leverage on all features**

Practical Effectiveness of Poisoning Attacks

Success Rate (SR): Percentage of attacks that are successful on the victim

SR of the StingRay poisoning attack on Drebin

Existing models can capture training set knowledge limitations

Practical attack success rate is overestimated by 36%

Hiding 90% of the training set decreases success rate by 50%

White-Box

Implicit Full Leverage

10% Known Training Instances

Practical Partial Leverage
Contributions

• FAIL adversarial model for highlighting realistic adversarial capabilities
  – Represents knowledge and control along: Features, Algorithms, Instances, Leverage

• StingRay, a generic targeted poisoning attack
  – Implemented on four applications and against three defenses

• Systematic evaluation of how much adversarial success depends on implicit assumptions
  – More accurate threat assessment
Outline

• FAIL
• StingRay
• Evaluation
The FAIL Model

• Models adversaries with variable levels of knowledge and capabilities across four dimensions:
  – Features
  – Algorithms
  – Instances
  – Leverage
FAIL in Action

Proprietary features from binaries (Program analysis)
Guessable training algorithm with unknown parameters (linear SVM)
Known and publicly accessible data set (Drebin data set)
Hard-to-modify features (Program analysis)

Partial Feature knowledge
Partial Algorithm knowledge
Full Instance knowledge
Partial Leverage

Generalized transferability measures the attacks success rate under realistic knowledge and capabilities assumptions
FAIL - Features

• Models the degree of knowledge about the adversarial features
  – What features can be kept secret?
  – Are the exact feature values known?

• Examples:
  – Unknown program analysis features
  – Unknown image resolution
FAIL - Algorithms

• Models the degree of knowledge about the classifier
  – Is the algorithm class known?
  – Is the training algorithm known?
  – Are the model parameters secret?

• Examples:
  – Unknown linear training algorithm
  – Unknown neural network architecture
FAIL – Instances

• Measures the overlap between the attack and the victim training sets
  – Is the entire training set public?
  – Are some instances known?
  – Are public instances sufficient to train a robust classifier?

• Examples:
  – Unknown malware training set
  – Public image training set
FAIL - Leverage

• Limits the crafting capabilities of the attacker
  – Which feature can be modified by the attacker?
  – Does the attack on some features have side effects?

• Examples:
  – Hard to modify program analysis features
  – Watermarked images
Outline

• FAIL
• StingRay
• Evaluation
Four Target Applications

• Drebin[1]: Android malware detector based on SVM
• Image classifier: Convolutional Neural Networks
• Twitter exploit predictor[7]: SVM classifier
• Breach predictor[8]: Random Forests on timeseries

The Poisoner’s Dilemma (1)

Victim Training Instances

Victim Decision Boundary

Instance labels cannot be assigned by the attacker
The Poisoner’s Dilemma (2)

Victim Training Instances

Target

Poisoning Instances

Detected outliers

Victim Decision Boundary

Poisoning instances could be detected by existing defenses
Poisoning instances could cause collateral, indiscriminate damage
StingRay achieves both individual and collective inconspicuousness.
Attack Requirements

• StingRay design requirements:
  – No assumed control over the labeling function
  – Individually and collectively inconspicuous poisoning
  – Practical FAIL considerations
StingRay High Level Illustration

- Target
- Base Instances
- Poisoning Instances
- Victim Training Instances
- Victim Testing Instances

Victim Decision Boundary
Poisoned Decision Boundary
Crafting Example - Drebin

api_call::setWifiEnabled
permission::WRITE_CONTACTS
permission.CALL_PHONE
permission::ACCESS_WIFI_STATE
permission::READ_CONTACTS
intent.action.SEARCH
intent.action.MAIN

VirusTotal highlights some features as more suspicious than others.
StingRay – Choosing a Base Instances

Choose base instances with some similarity to target

api_call::setWifiEnabled
permission::WRITE_CONTACTS
permission::CALL_PHONE
permission::ACCESS_WIFI_STATE
permission::READ_CONTACTS
intent.action::SEARCH
intent.action::MAIN

api_call::setWifiEnabled
permission::ACCESS_WIFI_STATE
activity::MainActivity
permission::READ_CONTACTS
StingRay – Individual Inconspicuousness

api_call::setWifiEnabled
permission::WRITE_CONTACTS
permission::CALL_PHONE
permission::ACCESS_WIFI_STATE
permission::READ_CONTACTS
intent.action.SEARCH
intent.action.MAIN

Reusing existing instances mitigates lack of leverage on some features
**StingRay – Collective Inconspicuousness**

<table>
<thead>
<tr>
<th>api_call::setWifiEnabled</th>
<th>permission::WRITE_CONTACTS</th>
<th>permission::ACCESS_WIFI_STATE</th>
<th>intent.action.SEARCH</th>
<th>intent.action.MAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>api_call::setWifiEnabled</td>
<td>permission::ACCESS_WIFI_STATE</td>
<td>activity::MainActivity</td>
<td>permission::READ_CONTACTS</td>
<td>permission::WRITE_CONTACTS</td>
</tr>
</tbody>
</table>

Poison instances bypass three defenses: RONI, targeted RONI and Micromodels.
### StingRay – Uncontrolled Labels

<table>
<thead>
<tr>
<th>api_call::setWifiEnabled</th>
<th>permission::WRITE_CONTACTS</th>
<th>permission.CALL_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>permission::ACCESS_WIFI_STATE</td>
<td>permission::READ_CONTACTS</td>
<td>intent.action.SEARCH</td>
</tr>
<tr>
<td>intent.action.MAIN</td>
<td>api_call::setWifiEnabled</td>
<td>permission::ACCESS_WIFI_STATE</td>
</tr>
<tr>
<td>permission::READ_CONTACTS</td>
<td>intent.action.MAIN</td>
<td>unknown label</td>
</tr>
</tbody>
</table>

89% of the 19,000 crafted apps are labeled as benign by VirusTotal

<table>
<thead>
<tr>
<th>api_call::setWifiEnabled</th>
<th>permission::ACCESS_WIFI_STATE</th>
<th>activity::MainActivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>permission::READ_CONTACTS</td>
<td>intent.action.SEARCH</td>
<td>unknown label</td>
</tr>
</tbody>
</table>

Crafting Example – Neural Networks

• Neural Networks learn features from raw data
• Adapting JSMA\[^6\] for poisoning
  – JSMA pushes instances towards class and not an instance
  – We modify JSMA’s objective function to move the poisoning instances towards the target

StingRay - White-Box Performance

Success Rate (SR): Percentage of attacks that are successful on the victim

Success Rate of StingRay in white-box setting

- Drebin
- Image Classifier
- Exploit Predictor
- Breach Predictor

Outline

• FAIL
• StingRay
• Evaluation
Some attacks perceived failed on the surrogate model are actually successful on the victim.

Feature secrecy appears to be the most powerful limiting factor.
**StingRay and JSMA - Success on the Image Classifier**

StingRay remains successful on all dimensions, sometimes even with increased efficiency due to a constrained localized strategy.

JSMA is more effective in white-box settings, but performs poorly on all other dimensions, in contrast to prior observations for Algorithms[9]

## FAIL Captures Adversaries from Prior Work

<table>
<thead>
<tr>
<th>Prior Work</th>
<th>F</th>
<th>A</th>
<th>I</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing-time attacks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FGSM Evasion (Goodfellow et al., 2014)</td>
<td>o</td>
<td>o</td>
<td>N/A</td>
<td>o</td>
</tr>
<tr>
<td>Model Stealing (Tramer et al., 2016)</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Genetic Evasion (Xu et al., 2016)</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Black-box Evasion (Papernot et al., 2017)</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>C&amp;W Evasion (Carlini et al., 2017)</td>
<td>o</td>
<td>o</td>
<td>N/A</td>
<td>o</td>
</tr>
<tr>
<td>Training-time attacks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVM Poisoning Attack (Biggio et al., 2012)</td>
<td>o</td>
<td>o</td>
<td>N/A</td>
<td>o</td>
</tr>
<tr>
<td>DNNs Poisoning (Munoz-Gonzalez et al., 2017)</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>NNs Backdoors (Gu et al., 2017)</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>NNs Trojaning (Liu et al., 2017)</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

- **Fully considered**
- **Considered, not evaluated**
- **Not considered**

Conclusions

• FAIL
  – Models realistic adversarial assumptions
  – Captures existing adversaries
  – Generalizes the notion of transferability
  – Applicable to both evasion and poisoning

• StingRay
  – Targeted poisoning attack
  – Crafts inconspicuous samples
  – No assumed control over the labeling function
  – Implemented against four applications
Thank you!

Octavian Suciu
osuciu.com
osuciu@umiacs.umd.edu

FAIL Framework available at:

ter.ps/fail