Intriguing Properties of Adversarial ML Attacks in the Problem Space

Fabio Pierazzi
Assistant Professor at King's College London, UK
<fabio.pierazzi@kcl.ac.uk>
https://fabio.pierazzi.com

Talk based on the IEEE S&P ’20 paper with Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro

SoSySec Seminar
IRISA, Rennes
June 19, 2020
A Dystopian Future...

Pandas are forbidden!
Guilty of being too cute!
Luckily, pandas are fluent in math…
Luckily, pandas are fluent in math…

Feature-space noise mask

“panda”
57.7% confidence

? =

“gibbon”
99.3% confidence
What happens in the **problem-space**, i.e., the real world?
What happens in the **problem-space**, i.e., the real world?
What happens in the **problem-space**, i.e., the real world?
Let’s Analyze What Happened

Feature-Space Attacks

Original Image  Perturbation  Adv. Image

“panda” 57.7%  imperceptible noise  “gibbon” 99.3%

\[ x + \delta \]
Let’s Analyze What Happened

Feature-Space Attacks

Original Image

"panda" 57.7%

Feature-Space Attacks

Perturbation

imperceptible noise

Adv. Image

"gibbon" 99.3%

Optimization

\[
\minimize_\delta \left( \| \delta \|_p + c \cdot f(x + \delta) \right)
\]

Pixel Perturbations

Loss of Target Class

\[ x + \delta \]
Let’s Analyze What Happened

Problem-Space Attacks

Feature-Space Attacks

Original Image

Perturbation

Adversarial Image

Original App (z)

"Perturbation"

Adversarial App (z’)

Feature Space

Problem Space

Optimization

minimize \( \| \delta \|_p + c \cdot f(x + \delta) \)

Constraints

- Is it realistic/plausible?
- Does it crash?
- Can it be detected by signatures?
- Does it preserve malicious functionality?
- … are there “general” constraints?

Problem-Space Attacks

Feature-Space Attacks

Original App (z)

“Perturbation”

Adversarial App (z’)

Original Image

Perturbation

Adversarial Image
Inverse Feature-Mapping Problem

The feature mapping $\varphi$ is differentiable — you can backpropagate to input.
Inverse Feature-Mapping Problem

In the software domain, the feature mapping $\varphi$ is neither invertible nor differentiable — how to get back to the problem space?
Many Problem-Space Attack Papers

- Android Malware
  [TDSC’17, ESORICS’17, ACSAC’19]
- Windows Malware
  [RAID’18, EUSIPCO’18]
- PDF Malware
  [ECML-PKDD’13, NDSS’16]
- Network Traffic
  [NCA’18, NCA’19]

What is the State of the Art? How to compare them?
Outline

Formalization

- Problem-space attacks
- Relationships
- Actionable points

Android Problem-Space Attack

- End-to-end adversarial malware generation at scale
- Feasible to evade feature-space defenses
Outline

Formalization
- Problem-space attacks
- Relationships
- Actionable points

Android Problem-Space Attack
- End-to-end adversarial malware generation at scale
- Feasible to evade feature-space defenses
Formalization
Background: Feature-Space Attacks

Test-time evasion

\[
\delta^* = \arg \min_{\delta \in \mathbb{R}^n} f_i(x + \delta), \quad \delta \in \Omega.
\]

- Optimal FS Perturbation
- Feature-space Constraints
- Loss of target class

Optimal FS

Perturbation

\( \delta^* \)

subject to:

\( \delta \in \Omega \).
Background: Feature-Space Attacks

Test-time evasion

\[ \delta^* = \arg \min_{\delta \in \mathbb{R}^n} f_t(x + \delta) \text{ subject to: } \delta \in \Omega. \]

Threat Models (Attacker Knowledge):

- **White box**: Feature Space, Algorithm, Training Data [1]
- **Gray box**
- **Black box**: None - perhaps type of Feature Space

Problem-Space Constraints
Problem-Space Constraints

Available Transformations

How can you alter problem-space objects?

- Addition
- Removal
- Modification

API
“strings”
bytes
Software
API
“strings”
bytes
Problem-Space Constraints

*Preserved Semantics*

*Available Transformations*

Which semantics do you preserve? How?
Which automatic tests can verify it?
Problem-Space Constraints

Preserved Semantics

Available Transformations

Test Suite
- Does it crash?
- Does it still communicate with CnC?
- Does it still encrypt the /home/ folder?

By Construction
- Add no-op operations
- Ensure it is not executed at runtime

Which semantics do you preserve? How?
Which automatic tests can verify it?
Problem-Space Constraints

- **Plausibility**
- **Preserved Semantics**
- **Available Transformations**

Test Suite
- User studies
- Automated heuristics

**By Construction**
- Taking precautions during mutation

Does it look legit?
Problem-Space Constraints

- Robustness to Preprocessing
- Plausibility
- Preserved Semantics
- Available Transformations

Which preprocessing are you considering?
Side-effect Features
Side-effect Features

Feature-Space Feasibility Region

Feature-Space

Feasibility Region

\( \Omega \)

\( x_1 \)

\( x_2 \)
Side-effect Features

[Graph showing feasibility regions in both feature space and problem space, with labels for Problem-Space Feasibility Regions and Feature-Space Feasibility Region.]
Side-effect Features

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure}
\end{figure}
Side-effect Features

\[ x + \delta^* \]
Side-effect Features

"Optimal" Feature-Space Perturbation
Side-effect Features

\[ x + \delta^* + \eta \]

Projection

"Optimal" Feature-Space Perturbation

\[ x + \delta^* \]
Side-effect Features
Side-effect Features

- Constraints:
  - PS Feasibility-Region $\subseteq$ FS Feasibility-Region

- Attack Confidence:
  - $a_{PS} \leq a_{FS}$
Actionable Points
Actionable Points

Verify existence of feature-space attack

Necessary Condition for problem-space attacks

∃ problem-space attack $\implies$ ∃ feature-space attack

Proof 1
in paper
**Actionable Points**

Verify existence of feature-space attack

**Necessary Condition for problem-space attacks**

\[ \exists \text{ problem-space attack} \implies \exists \text{ feature-space attack} \]

Proof 1 in paper

Identify approximate inverse feature mapping

**Sufficient Condition for problem-space attacks**

\[ \exists \text{ problem-space attack} \iff \exists \text{ feature-space attack}, \exists \text{ approximate } \phi^{-1} \]

Proof 2 in paper
Search Strategy

Problem-driven vs. Feature-Driven

Modify problem-space object

- Add API
- Add Strings
- Add Bytes

Repeat until it evades the classifier.
Feature Space vs. Problem Space

Feature-Space Constraints
- Lp perturbations
- Domain constraints for vectors

Search Strategy
- Feature-driven

Problem-Space Constraints
- Available Transformations
- Preserved Semantics
- Plausibility
- Robustness to Preprocessing

Search Strategy
- Feature-driven
- Problem-driven
- Hybrid

\[
\delta^* = \arg \min_{\delta \in \mathbb{R}^n} f_t(x + \delta)
\]
subject to: \( \delta \models \Omega \).

\[
\arg\min_{T \in \mathcal{T}} f_t(\varphi(T(z))) = f_t(x + \delta^* + \eta)
\]
subject to:
\[
\begin{align*}
[z]^r &= [T(z)]^r, \quad \forall \tau \in \mathcal{Y} \\
\pi(T(z)) &= 1, \quad \forall \pi \in \Pi \\
A(T(z)) &= T(z), \quad \forall A \in \Lambda
\end{align*}
\]
Feature Space vs. Problem Space

\[ \delta^* = \arg \min_{\delta \in \mathbb{R}^n} f_t(x + \delta) \]
subject to: \[ \delta \models \Omega. \]

Feature-Space Constraints
- Lp perturbations
- Domain constraints for vectors

Search Strategy
- Feature-driven

Problem-Space Constraints
- Available Transformations
- Preserved Semantics
- Plausibility
- Robustness to Preprocessing

Search Strategy
- Feature-driven
- Problem-driven
- Hybrid
Works for Multiple Domains

Compare existing methods & improve SoA

Available Transformations
Preserved Semantics
Plausibility
Robustness to Preprocessing

See Table I in §II.C
Works for Multiple Domains

Compare existing methods & improve SoA

See Table I in §II.C

Available Transformations

Preserved Semantics

Plausibility

Robustness to Preprocessing

Facial Recognition

Speech Recognition

Android Malware

Malicious Javascript

PDF Malware

Image Classification

Text Classification

Windows Malware

Code Attribution

Fabio Pierazzi (KCL) - *Intriguing Properties of Adversarial ML Attacks in the Problem Space* - SoSySec Seminar, IRISA, Rennes - June 19, 2020
Android Attack
Prior Work on Adv. Malware

Prior work was fundamental to initially explore problem-space attacks. We propose a principled approach that supports reasoning.

Available Transformations
• Limiting #features modified

Robustness to Preprocessing
• Removable unused permissions
• Removal code (unreachable, no-op)
• Unclear

Preserved Semantics
• Highly unstable transformations
Our Android Attack
Our Android Attack
Our Android Attack

Available Transformations
Code addition through automated software transplantation.
Our Android Attack

- **Available Transformations**
  Code addition through automated software transplantation.

- **Preserved Semantics**
  Malicious semantics preserved by construction using opaque predicates (inserted code is not executed at runtime).
Our Android Attack

**Available Transformations**
Code addition through automated software transplantation.

**Preserved Semantics**
Malicious semantics preserved by construction using opaque predicates (inserted code is not executed at runtime).

**Robustness to Preprocessing**
We’re robust to:
- removal of redundant code
- undeclared variables
- unlinked resources
- undefined references
- naming conflicts
- no-op instructions.
Our Android Attack

Available Transformations
Code addition through automated software transplantation.

Preserved Semantics
Malicious semantics preserved by construction using opaque predicates (inserted code is not executed at runtime).

Robustness to Preprocessing
We’re robust to:
- removal of redundant code
- undeclared variables
- unlinked resources
- undefined references
- naming conflicts
- no-op instructions.

Plausibility
Only realistic code is injected (rather than orphaned urls, api calls, etc.)
Mutated apps install and start on an emulator.
Organ Harvesting
Organ Harvesting

1. Identify feature entry point

Identify activity in dex
Organ Harvesting

2. Choose any vein (backward slice)

Extract intent creation and startActivity()
Organ Harvesting

3. Collect organ (forward slice)

Gather activity definition
Organ Harvesting

4. Include transitive dependencies

Recursively collect dependencies
Organ Harvesting

5 Store organ in an “ice box”

Save gadget to a database ready for the attack
Attack Overview
Attack Overview

Given a trained target model
Attack Overview

Given a trained target model

First pick feature with greatest ‘benign’ weight
Attack Overview

Given a trained target model
First pick feature with greatest ‘benign’ weight
Find a corresponding organ from the ice box
Attack Overview

Given a trained target model
First pick feature with greatest ‘benign’ weight
Find a corresponding organ from the ice box
Wrap the organ in an opaque predicate
Attack Overview

Given a trained target model
First pick feature with greatest ‘benign’ weight
Find a corresponding organ from the ice box
Wrap the organ in an opaque predicate
Inject the new benign code and repackage
Opaque Predicates

- Example of opaque predicate in **JSketch**
  - Opaque predicate wraps an *adapted vein*
  - Random k-SAT parameters

```java
void opaque() {
    Random random = new Random();
    this();
    boolean[] arrayOfBoolean = new boolean[40];
    byte b1;
    for (b1 = 0; b1 < arrayOfBoolean.length; b1++)
        arrayOfBoolean[b1] = random.nextBoolean();
    b1 = 1;
    for (byte b2 = 0; b2 < 184.0D; b2++) {
        boolean bool = false;
        for (byte b = 0; b < 3; b++)
            bool |= arrayOfBoolean[random.nextInt(
                arrayOfBoolean.length)];
        if (!bool)
            b1 = 0;
    }
    if (b1 != 0) {
        // Beginning of adapted vein
        Context context = ((Context)this).getApplicationContext();
        Intent intent = new Intent();
        this(this, h.a(this, exim.qngg.TEhr.sFiQa.class));
        intent.putExtra("1", h.p(this));
        intent.addFlags(268435456);
        startActivityForResult(intent);
        h.x(this);
        return;
        // End of adapted vein
    }
}
```
Attack Overview

Continue choosing benign features until the app is misclassified
Side-Effects
Side-Effects

Each organ contains side-effect features.
Side-Effects

Each organ contains side-effect features.

We can sum target features, positive, and negative side effects.
Side-Effects

Each organ contains side-effect features. We can sum target features, positive, and negative side effects to choose organs in order of their overall benign weight.
Android Attack: Experiments
Experimental Testbed

Dataset

• ~170K Android apps (10% malware) from Jan 2017 to Dec 2018
• 66% training - 34% testing (random split, to remove “concept drift” as a variable)

[Jordaney et al., “TRANSCEND”, USENIX Sec 2017; Pendlebury et al., “TESSERACT”, USENIX Sec 2019]
Experimental Testbed

Dataset

- ~170K Android apps (10% malware) from Jan 2017 to Dec 2018
- 66% training - 34% testing (random split, to remove “concept drift” as a variable)

\[\text{Jordaney et al., “TRANSCEND”, USENIX Sec 2017; Pendlebury et al., “TESSERACT”, USENIX Sec 2019}\]

Detection algorithms

- DREBIN [NDSS’14]: Linear SVM, binary feature space
- Sec-SVM [TDSC’17]: Feature-space defense for DREBIN (uses “more evenly-distributed weights”)
Experimental Testbed

Dataset

- ~170K Android apps (10% malware) from Jan 2017 to Dec 2018
- 66% training - 34% testing (random split, to remove “concept drift” as a variable)

[Jordaney et al., "TRANSCEND", USENIX Sec 2017; Pendlebury et al., "TESSERACT", USENIX Sec 2019]

Detection algorithms

- DREBIN [NDSS’14]: Linear SVM, binary feature space
- Sec-SVM [TDSC’17]: Feature-space defense for DREBIN (uses “more evenly-distributed weights”)

Attack Configurations

- Low Confidence (L): overcome decision boundary
- High Confidence (H): first quartile of benign distribution
Results

• Attack Success Rate: 100%
  › ~15K adversarial malware in total
Results

• Attack Success Rate: 100%
  › ~15K adversarial malware in total

• Experimental Questions
  › We do not limit the feature-space perturbation: what is the feature-space impact?
    › e.g., in images or audio there is a point in minimizing the perturbation
  › How much are the **app statistics** affected?
  › Is the attack practical? How much **time** does it take?
Results: What is the impact on Feature Space?
Results: What is the impact on Feature Space?

- Perturbations include side-effect features
- Sec-SVM (feature-space defense) forces the attacker to modify more features
  - Security-Performance trade-off
- Next slides: Does adding many features affect significantly app statistics?
Results: How much are app statistics affected?

- Adding all these features (+ side-effect features), what does it do to app statistics?
  - Limiting feature-space perturbations $\delta$ does not affect problem-space attack
Results: How much are app statistics affected?

- Adding all these features (+ side-effect features), what does it do to app statistics?
  - Limiting feature-space perturbations $\delta$ does not affect problem-space attack
Results: How much are app statistics affected?

- Adding all these features (+ side-effect features), what does it do to app statistics?
  - Limiting feature-space perturbations $\delta$ does not affect problem-space attack

![Diagram showing successful evasive apps percentage against application size]
Results: How much are app statistics affected?

- Adding all these features (+ side-effect features), what does it do to app statistics?
  - Limiting feature-space perturbations $\delta$ does not affect problem-space attack
Results: How much time does an attack take?

- In most cases, **less than 2 minutes** to create an adversarial example
Conclusions
Problem-Space Adversarial ML Attacks

• Novel reformulation of adversarial attacks
• Novel end-to-end adversarial malware generation

Project website (with code):
• https://s2lab.kcl.ac.uk/projects/intriguing/

Problem-Space Constraints
• Available Transformations
• Preserved Semantics
• Plausibility
• Robustness to Preprocessing

Search Strategies
• Gradient-driven
• Problem-driven
• Hybrid

Attack Code: Publicly available

- Project website: https://s2lab.kcl.ac.uk/projects/intriguing/
  - Attack Code and Dataset (released May 1, 2020)
  - Available to Researchers under MIT license
Problem-Space Adversarial ML Attacks

- Novel reformulation of adversarial attacks
- Novel end-to-end adversarial malware generation

Project website (with code):
- https://s2lab.kcl.ac.uk/projects/intriguing/

Problem-space attacks research is just beginning!

Problem-Space Constraints
- Available Transformations
- Preserved Semantics
- Plausibility
- Robustness to Preprocessing

Search Strategies
- Gradient-driven
- Problem-driven
- Hybrid