Certifiably robust malware detectors by design

Pierre-François Gimenez, Inria
Sarath Sivaprasad, CISPA Helmholtz Center for Information Security
Mario Fritz, CISPA Helmholtz Center for Information Security

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

IFIP SEC, May 23rd, 2025

Inria

Context

Malware

A malware is a malicious software: botnet, encryption, backdoor, cryptocurrency mining...

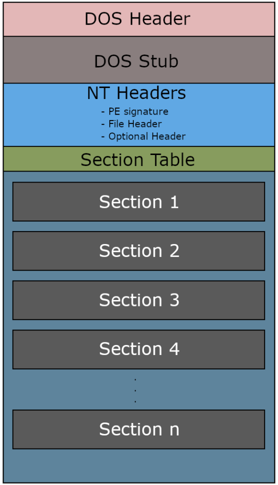
Malware analysis

Two main categories of malware analysis:

- static analysis, where the software is not run
  - extracted features: control flow graph, file metadata, library imports, presence of encryption, etc.
- dynamic analysis, where the software is monitored during its execution
  - extracted features: network activity, modified files, system calls list, etc.

These features can be used by machine learning to help detect, classify and cluster malware

Windows executable file



Our work

- We focus on **Windows malware**, the most common desktop target
- We study **static analysis** for its ease of experiment and scaling capability

PE format

- Legacy content for backward compatibility (DOS header and DOS stub, etc.)
- Flexible format: the order of the sections is free, some parts are optional, etc.

Adversarial attacks

Attacks on machine learning

- Machine learning is increasingly used to analyze malware
- Many attacks against machine learning, at different stages (data collection, learning, inference) and targeting different properties (integrity, privacy, etc.)

Evasion attacks

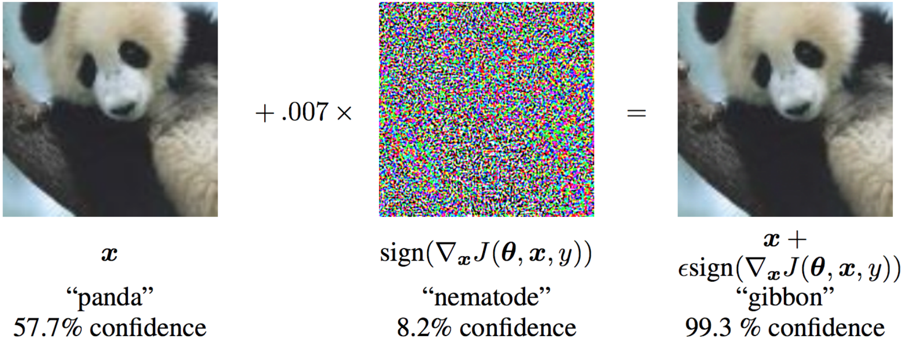
- The goal of the attacker is to modify slightly the features to change the predicted class
- Formally, for an input $x \in \mathbb{R}^n$, the attacker looks for a “small” $\epsilon \in \mathbb{R}^n$ such as $argmax_c f_c(x) \neq argmax_c f_c(x + \epsilon)$ (i.e., the predicted class changed)

Question: how to make malware classifiers more robust?

Adversarial examples against malware detectors

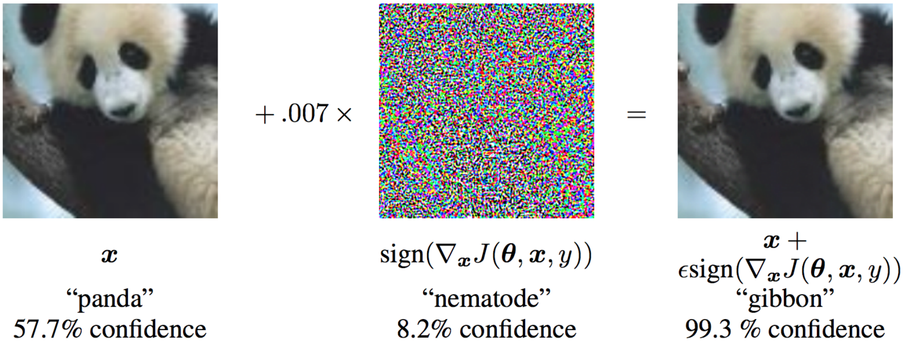
The issue

Even very accurate classifiers can be fooled by slightly modifying the input



The issue

Even very accurate classifiers can be fooled by slightly modifying the input



What about malware?

Adversarial examples

Image $\neq$ malware

- We cannot randomly modify a malware and expect it to work correctly
- Images are continuous: small variations do not change their meaning
- Programs are discrete: opcode "0x60" is very different from opcode "0x61"
- Perturbations on images must stay small to be invisible to human eyes
- Perturbations on programs do not have this constraint

$\Rightarrow$ the threat model is very different

Adversarial attacks on malware detection

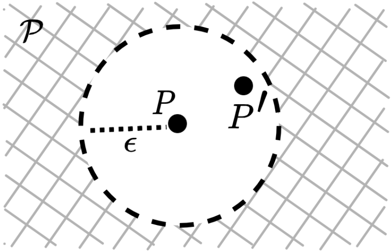
How to attack malware detectors

Most common approach: modify the malware with semantics-preserving operations:

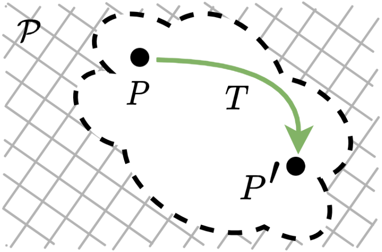
- file padding
- header perturbation
- API import addition
- ... and many more

Adversarial examples are build by chaining such operations in a black-box way

Detection evasion

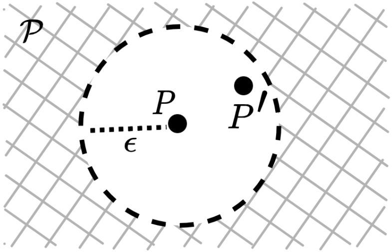


Attack on **images**. The attacker looks for an image within a ϵ-ball

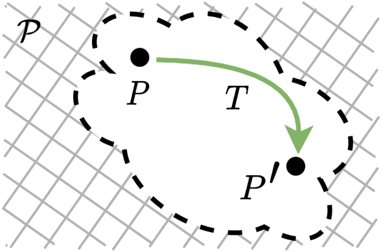


Attack on **malware**. P' must have the same behavior as P

Detection evasion



Attack on **images**. The attacker looks for an image within a ϵ-ball



Attack on **malware**. P' must have the same behavior as P

Current techniques against adversarial attacks assume the perturbation is small

This assumption is not reasonable for malware!

Certifiable robustness by design

Certifiably robust detector by design

Related work

- Prior work: one should only use features that cannot be decreased by transformations, along with a monotonic classifier
- Intuition: whatever the attacker does, the output of the classifier can only increase, i.e., the detector can only be more confident it is a malware
- If the assumption holds, then the classifier is robust: no attack is possible, no matter how large the perturbation is
- Accuracy results are underwhelming because many features are discarded

Intuition of our contribution

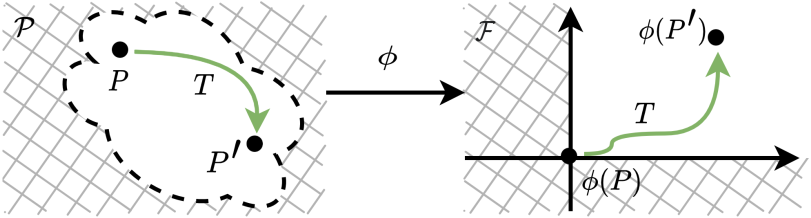
What about a more complex feature mapping?

- Earlier, the feature mapping is just a projection (keep or drop features)
- We could use adversarial examples to automatically learn the feature mapping ϕ
- That way, we could have much more expressive robust classifiers

Intuition of our contribution

What about a more complex feature mapping?

- Earlier, the feature mapping is just a projection (keep or drop features)
- We could use adversarial examples to automatically learn the feature mapping ϕ
- That way, we could have much more expressive robust classifiers



The feature mapping ϕ ensures that perturbations can only increases features

Expressivity

How expressive is it?

- Is this just a "hack"? Or a more profound insight into robust classifiers?
- We prove that **all robust classifiers can be expressed as a features mapping followed by a monotonic classifier**

Proposition

Let ϕ be a feature mapping and f be a classifier such that f is robust against adversarial attacks. There exist g and h such that $f \circ \phi = (f \circ h) \circ (g \circ \phi)$ and $f \circ h$ is monotonically increasing.

Example

Our proposition: learn the feature mapping

- Consider the attack that replaces one API call with a similar one (replacing `CreateFileA` with `CreateFileW`)
- This transformation modifies features f_1 (number of `CreateFileW`) and f_2 (number of `CreateFileA`) such as $f_1 \leftarrow f_1 + 1$ and $f_2 \leftarrow f_2 - 1$
- The previous work would drop f_2 (it can be decreased)
- If the other transformation (`CreateFileW` into `CreateFileA`) is possible, then f_1 would also be dropped!
- Our model could create the feature $f_3 = f_1 + f_2$ (number of `CreateFileA` and `CreateFileW`) and not lose as much information while still ensuring monotonicity

How to do that?

ERDALT

- We showed that every robust classifier can be structured as a monotonic classifier on top of some specially crafted feature mapping
- We propose to learn a neural network with two parts:
  - a first layer for the role of feature mapping
  - monotonic layers for the role of the detection
- We can prove, under some assumption, that this model is robust (by design)

We name our approach ERDALT: *Empirically Robust by Design with Adversarial Linear Transformation*

Properties

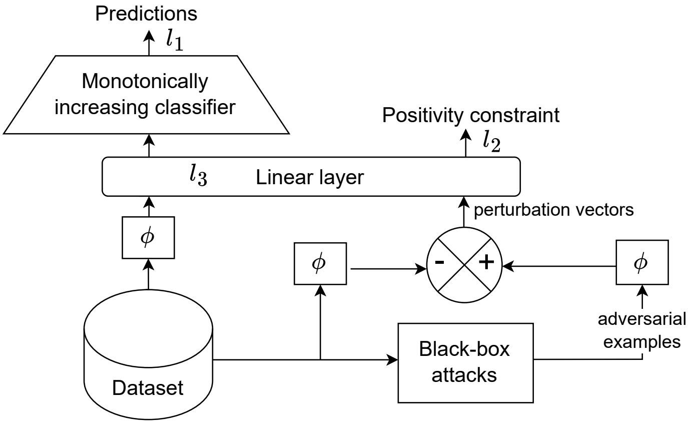
Assumption

- Without any assumption, we cannot hope to learn robust classifiers
- To obtain theoretical guarantees, we assume the effect of the transformations on the features is independent from the initial malware
  - Replacing an API call with a similar one decreases a feature and increments another
  - A padding transformation adds n bytes to a section

Linear feature mapping

- A linear feature mapping ensures that the effect of two transformations on the features is simply the sum of their effects
- If the model is robust against all elementary transformations, then it is robust against any combination of transformations!

ERDALT: empirically robust by design malware detector



ERDALT

- We want to minimise the classification error (loss l_1)
- The first linear layer maps perturbations vectors to positive values (loss l_2)
- A third loss encourages a sparse linear layer (loss l_3)

Experimental assessment

Experimental protocol

Dataset and features

- Dataset: created by EURECOM and Avast, contains 60,000 malware
- Features:
  - EMBER (state-of-the-art): 1871 features
  - Manually selected features: 40 features selected to be difficult to decrease

Adversarial attacks

- `secml-malware`
- Applies semantics-preserving transformations with a genetic algorithm

Metrics

- Performances are evaluated with ROC AUC
- Robustness: proportion of malware not successfully attacked

Performance with no protections

Model	**Manual features** (40)		**EMBER** (1871)	
	ROC AUC	**Robustness**	**ROC AUC**	**Robustness**
Baseline network	89.9%	**100%**	91.6%	**82.0%**
Monotonic network	69.0%	**100%**	87.4%	71.5%
Random Forest	**94.6%**	98.5%	96.2%	81.0%
AdaBoost	85.0%	98.0%	94.2%	75.5%
k-nn	83.7%	93.5%	88.6%	0%
Decision tree	84.1%	99.5%	96.2%	67.0%
Monotonic GBT	76.2%	**100%**	92.7%	73.5%
GBT	92.3%	99.0%	**97.5%**	75.0%

- Feature sets impact a lot the AUROC and robustness
- Manually selected features lead to much higher robustness and limited ROC AUC loss
- **More features means larger attack surface**

Performances with protections

Protection	Model	EMBER	
		ROC AUC	**Robustness**
Increasing-only features	Random Forest	95.2%	**100%**
	Monotonic GBT	86.7%	**100%**
	Gradient-boosted trees	93.8%	**100%**
Adversarial training	Random Forest	**97.6%**	94.5%
	Monotonic GBT	92.7%	95.5%
	Gradient-boosted trees	**97.6%**	96.5%
ERDALT	Neural network	93.0%	96.0%
ERDALT + adv. training	Neural network	85.5%	**100%**

Adversarial training yields the best ROC AUC, but the lowest robustness

Features selections

	Increasing-only features	ERDALT selection	Intersection
Byte	0%	84.9%	0%
Strings	1.9%	94.2%	1.9%
General	30.0%	60.0%	30.0%
Header	77.4%	83.9%	64.5%
Section	55.2%	76.5%	40.8%
Imports	44.5%	66.5%	22.2%
Exports	100%	49.2%	49.2%
Data directories	46.7%	90.0%	43.3%

ERDALT can exploit more features than the previous method due to the linear combinations it allows

Ablation study

Ablation study

- A typical ML experiment to analyze the effect of each component
- We can conclude that both the linear layer and the monotonicity are necessary for high robustness

Linear layer	**Monotonicity**	**ROC AUC**	**Robustness**
×	×	91.6%	82.0%
✓	×	**94.3%**	91.0%
×	✓	87.4%	71.5%
✓	✓	93.0%	**96.0%**

Conclusion

Adversarial attacks against malware detectors

- Attacks on images $\neq$ attacks on malware
- Provably robust methods assume the perturbation is small
- Our provably robust method does not rely on this unrealistic assumption

How to make a robust detector?

- Use a monotonic model with increasing features but expect a large performance drop
- Use ERDALT, which learns a feature mapping, and expect a smaller performance drop
- ERDALT can be combined with adversarial training as well

Perspectives

- Deep learning is not adapted to malware analysis
- We plan to apply this method to other security-related domains