



# Malware Detection in the AI Era: Attacks and Defenses on Machine Learning Classifiers

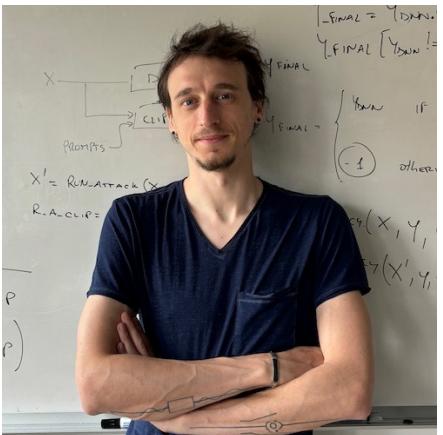
Luca Demetrio

Assistant Professor | University of Genova

Dmitrijs Trizna

Security Researcher | Microsoft Corporation, University of Genova

# WhoAreWe



Assistant Professor @ UNIGE

Doing research for fun and profit since 2019

Main contribution in offensive security  
with Adversarial EXEmples

Reviewer for top-tier journals and conferences

Working in EU projects



**Security Researcher @ Microsoft**

Tech-lead in M365/Azure back-end:  
AI/ML for Linux & Kubernetes cyber-threat  
detection

**Doctoral Researcher @ UNIGE**

AI/ML based malware detectors

In Past:

- Red Teaming
- NATO events
- OSCP, SANS/GIAC certs
- ...

# Agenda

## Part 3: Attacking ML-based Malware Detection

1. Introduction to Adversarial Machine Learning
2. Adversarial EXEmples: pentesting ML Windows Malware Detectors
3. Defenses against Adversarial EXEmples

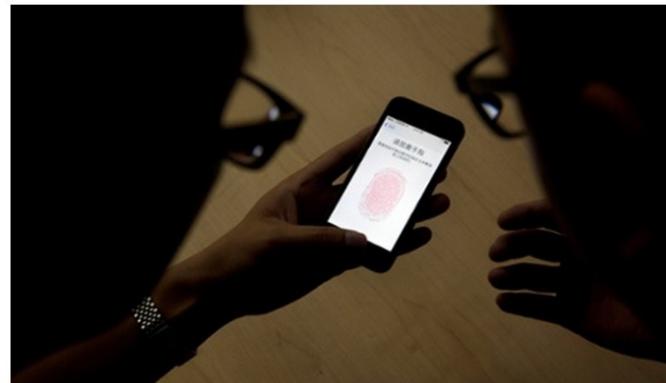
# ML is not bulletproof

iPhone 5S fingerprint sensor hacked by Germany's Chaos Computer Club

Biometrics are not safe, says famous hacker team who provide video showing how they could use a fake fingerprint to bypass phone's security lockscreen

[Follow Charles Arthur by email](#) BETA

Charles Arthur  
[theguardian.com](#), Monday 23 September 2013 08.50 BST  
[Jump to comments \(306\)](#)



While ML models possess tremendous performance, they can be target of attacks from anybody with some background in ML and reverse engineering

[Thanks to Prof.Roli, Prof. Biggio for this slide]

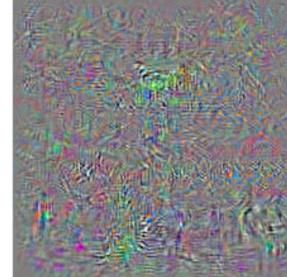
# Adversarial Examples

input image



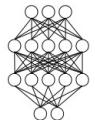
$+\epsilon$

adversarial perturbation



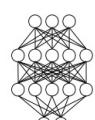
=

adversarial example



school bus (94%)

Mostly-invisible noise that force  
the model in predicting



ostrich (97%)

[Thanks to Prof.Roli, Prof. Biggio for this slide]

Szegedy et al., [Intriguing properties of neural networks](#), ICLR 2014  
Biggio et al., [Evasion attacks against machine learning at test time](#), ECML-PKDD 2013

# Attacks in the real world



[Thanks to Prof.Roli, Prof. Biggio for this slide]

Eykholt et al., *Robust physical-world attacks on deep learning visual classification*, CVPR 2018

# Stealing identities

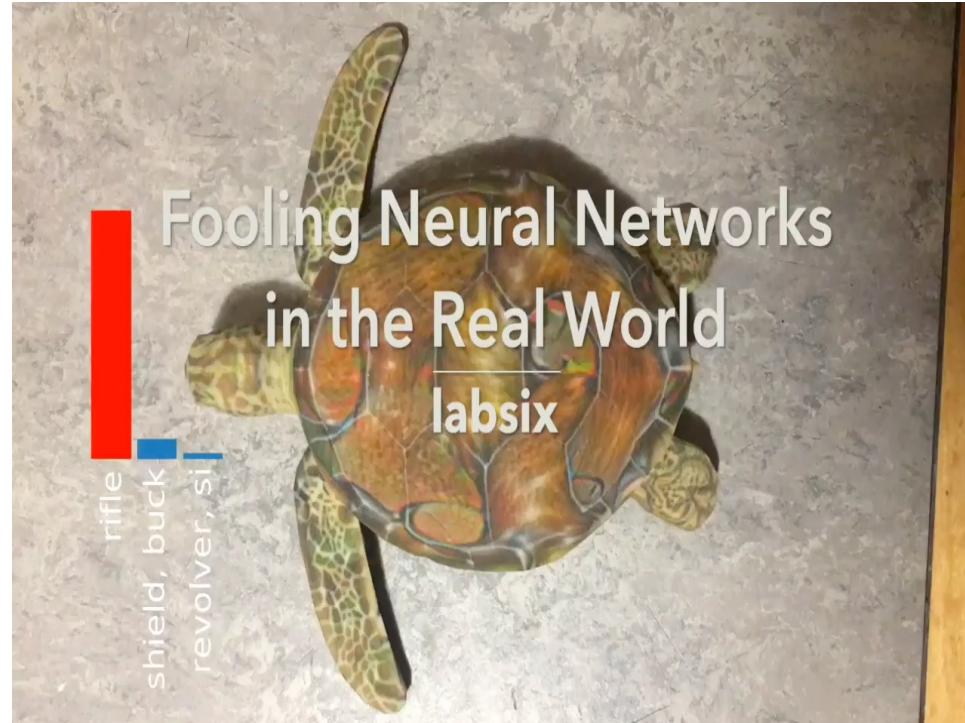
Attacks against DNNs for face recognition with carefully-fabricated eyeglass frames

When worn by a **41-year-old white male** (left image), the glasses mislead the deep network into believing that the face belongs to the famous actress **Milla Jovovich**



Sharif et al., *Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition*, ACM CCS 2016

# Printed objects detected as Rifles



[Thanks to Prof.Roli, Prof. Biggio for this slide]

Athalye et al., *Synthesizing robust adversarial examples*. ICLR, 2018

# Adversarial Audios

## Audio

### Transcription by Mozilla DeepSpeech



“without the dataset the article is useless”



“okay google browse to evil dot com”

Adding almost-inaudible noise to  
force speech-to-text models to  
capture the wrong words

[https://nicholas.carlini.com/code/audio\\_adversarial\\_examples/](https://nicholas.carlini.com/code/audio_adversarial_examples/)

[Thanks to Prof.Roli, Prof. Biggio for this slide]

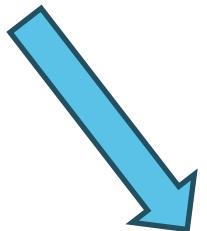
# Why all these insecurities?

# Spurious Correlations

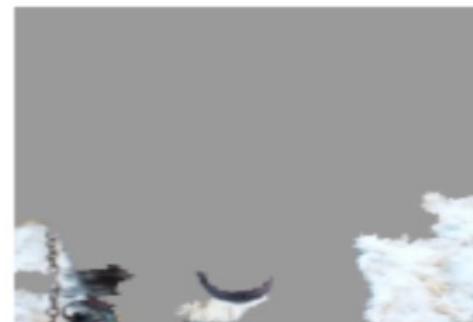


(a) Husky classified as wolf

“Why Wolf?”



“White background!”

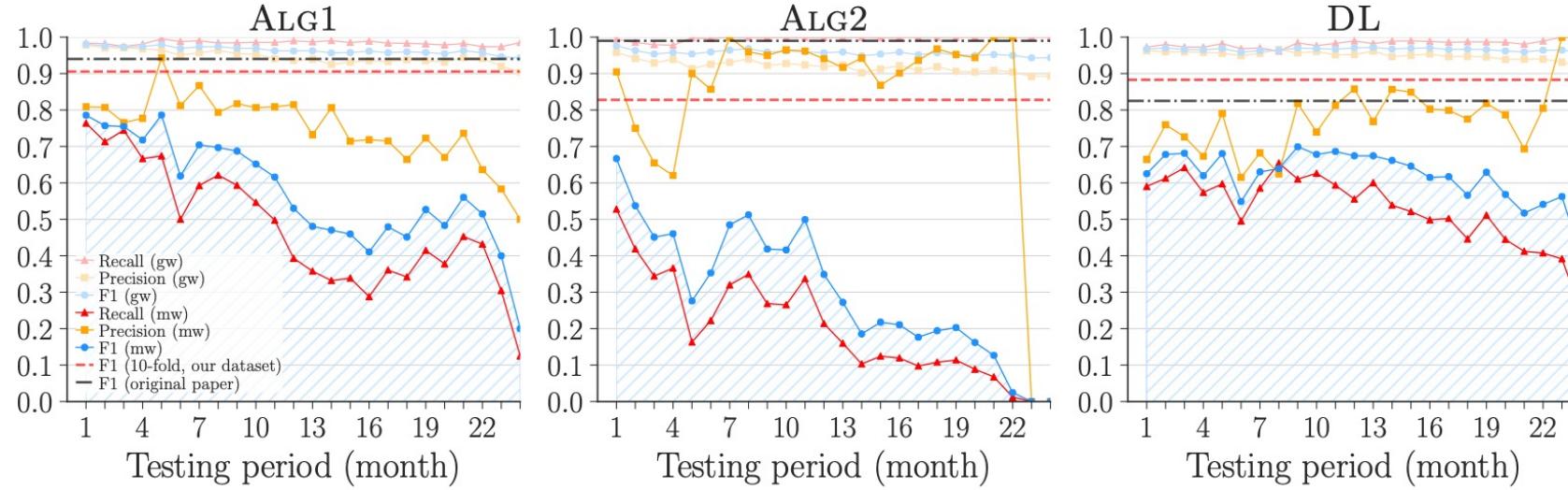


(b) Explanation

Models do not guarantee that they are learning what we believe it's important, but they just match good correlations in data

[M.T. Ribeiro et al., KDD 2016]

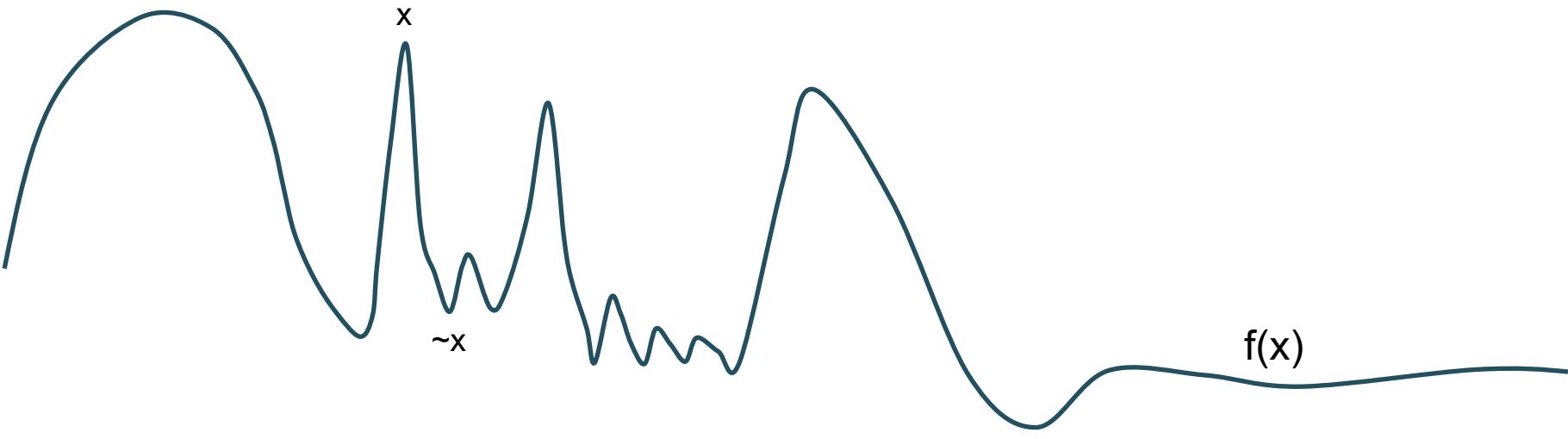
# Data is not stationary



Data changes over time, and  
models are trained with the IID  
assumption

Pendlebury, Cavallaro et al., TESSERACT, USENIX Sec. 2019

# Overfitting



If badly trained, ML models adapt **too much** on training data, becoming mostly useless at test time in presence of minimal noise

# Model Adversaries for profit and profit

# Anticipate the Arm Race



Instead of waiting for attacks and brace for the impact, system designers should **ALWAYS** simulate attacks and develop countermeasures

System designers should test strong attacks to understand the possible damages that adversaries might do!

# Goals, Knowledge and Capabilities

Formalize attackers through  
three different dimensions

## GOAL

The security violation that the attacker  
wants to achieve

## KNOWLEDGE

What the attacker knows about the  
system to test

## CAPABILITIES

Which action can be performed by the  
attacker

Biggio and Roli, **Wild Patterns: Ten Years After The Rise of Adversarial Machine Learning**, Pattern Recognition, 2018

# Attacker's Goal

## Integrity Violations

Misclassifications that do not compromise the normal system operation

## Availability Violations

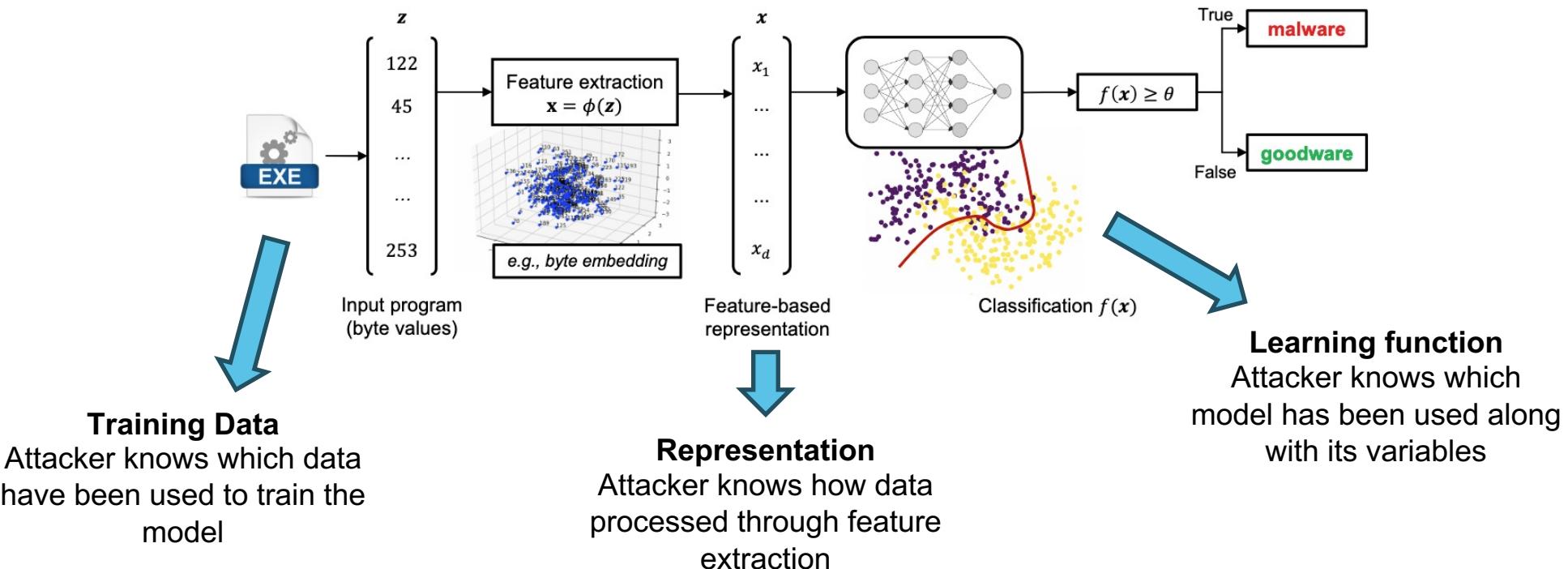
Misclassifications that compromise the normal system operation, causing a **Denial of Service** for all the users of that model

## Privacy / Confidentiality Violations

Strategies that leak confidential information about the model, the data used at training time, or stealing industrial and intellectual properties

Biggio and Roli, **Wild Patterns: Ten Years After The Rise of Adversarial Machine Learning**, Pattern Recognition, 2018

# Attacker's Knowledge



Biggio and Roli, **Wild Patterns: Ten Years After The Rise of Adversarial Machine Learning**, Pattern Recognition, 2018

# Perfect Knowledge

## Training Data

Attacker knows which data have been used to train the model



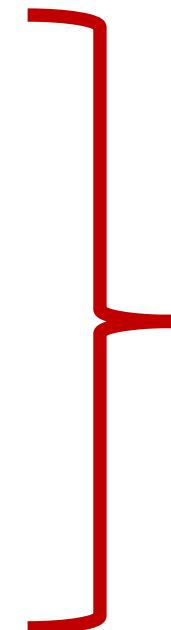
## Representation

Attacker knows how data processed through feature extraction



## Learning function

Attacker knows which model has been used along with its variables



## Perfect knowledge

Worst-case nightmarish scenario

The attacker knows **EVERYTHING** about our system  
Also known as **white-box access**

# Limited Knowledge

## Training Data

Attacker knows which data have been used to train the model



## Representation

Attacker knows how data processed through feature extraction



## Learning function

Attacker knows which model has been used along with its variables



**Limited knowledge**  
The attacker knows SOMETHING about our system, that could be a combination of the three listed assets, **but not all of them!**

Also known as **gray-box access**

# Zero Knowledge

## Training Data

Attacker knows which data have been used to train the model



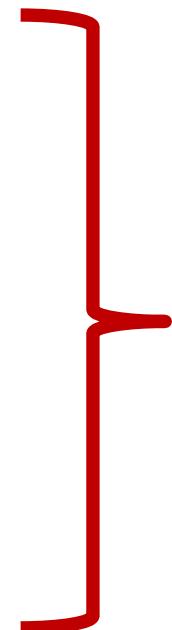
## Representation

Attacker knows how data processed through feature extraction



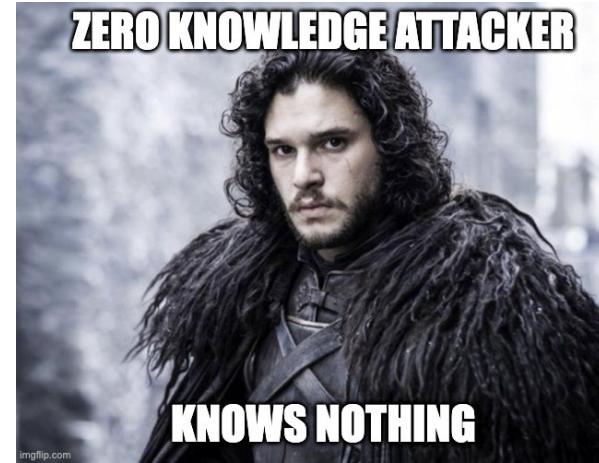
## Learning function

Attacker knows which model has been used along with its variables



## Zero knowledge

The attacker knows **NOTHING** about our system  
Also known as **black-box access**



# Attacker's Capabilities

## Test Time

The attacker can interact with the model AFTER deployment  
(at test time)

## Training Time

The attacker can tamper, manipulate, and interact with the model BEFORE deployment and while it is training.

Biggio and Roli, **Wild Patterns: Ten Years After The Rise of Adversarial Machine Learning**, Pattern Recognition, 2018

# Strength of the attack

## Test Time

The attacker can perturb data,  
but without infinite power

## Training Time

The attacker can tamper only  
with a fraction of the training data

Biggio and Roli, **Wild Patterns: Ten Years After The Rise of Adversarial Machine Learning**, Pattern Recognition, 2018

# Recap on modelling attackers

Attacker's Goal			
Attacker's Capability	Integrity	Availability	Privacy / Confidentiality
Test data	Misclassifications that do not compromise normal system operation	Misclassifications that compromise normal system operation	Querying strategies that reveal confidential information on the learning model or its users
Training data	Evasion (a.k.a. adversarial examples)	-	Model extraction / stealing Model inversion (hill-climbing) Membership inference attacks

All possible attacks can be recast to this table, making it compact and easy to use in production environments

Biggio and Roli, **Wild Patterns: Ten Years After The Rise of Adversarial Machine Learning**, Pattern Recognition, 2018

# Evasion attacks with Adversarial Examples

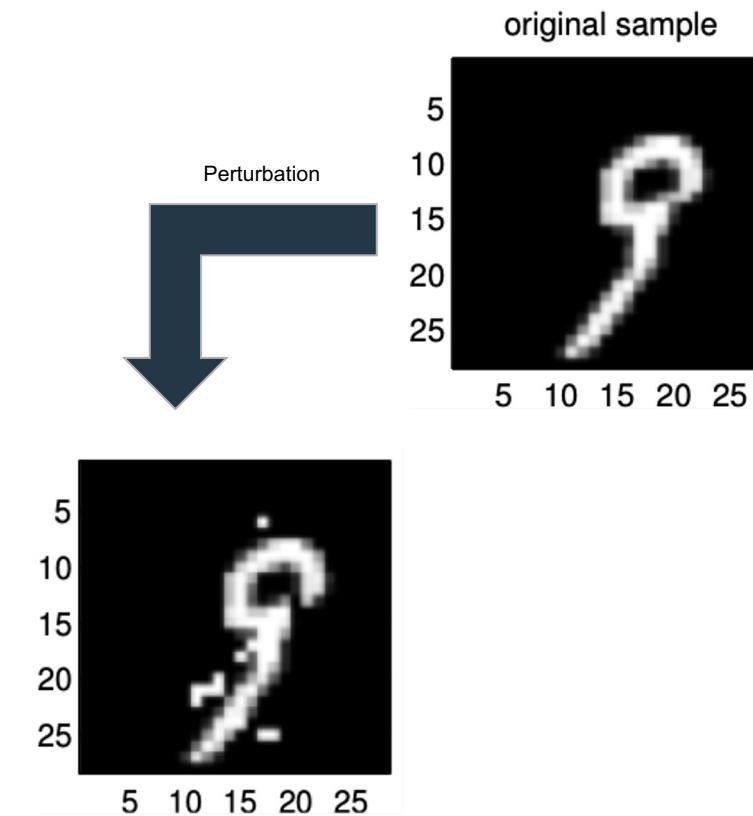
# Adversarial Examples

## Perturbed test samples

Modifying values of pixels that maximise the error at test time

## Constraint perturbation

Samples are minimally altered so that they do not look suspicious



# Formalization of the attack

**Maximisation**  
we want to increase  
the error of the  
model

**Perturbation**  
what is injected  
in the test point  
 $x$

**Parameters of the model**  
included inside the loss  
function

$$\max_{\delta} L(x + \delta, y; \theta)$$

**Loss function**  
distance function  
used to quantify the  
error

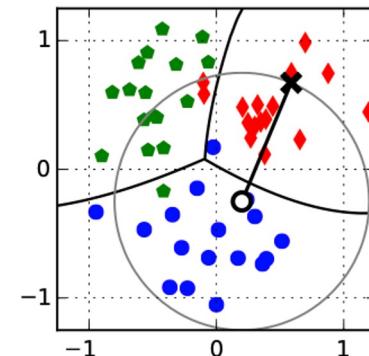
**Real label**  
the label from  
which we  
want to get  
away

# Error-generic (untargeted) attacks

$$\max_{\delta} L(x + \delta, y; \theta)$$

## Maximum error at test time

The perturbation is crafted to redirect the decision towards any other class different from the original one

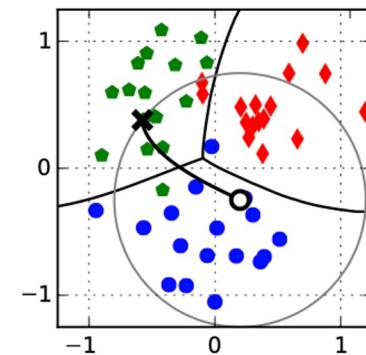


# Error-specific (targeted) attacks

$$\min_{\delta} L(x + \delta, y_t; \theta)$$

**Minimize error of target class**

Change from MAX to MIN, force the perturbation to redirect decision towards a specific class



# Including the Capabilities

$$\max_{\delta} L(x+\delta, y; \theta)$$

$$\text{s.t. } \|\delta\|_p < \epsilon$$

and

$$l_b \leq x + \delta \leq u_b$$

## Bounded perturbation

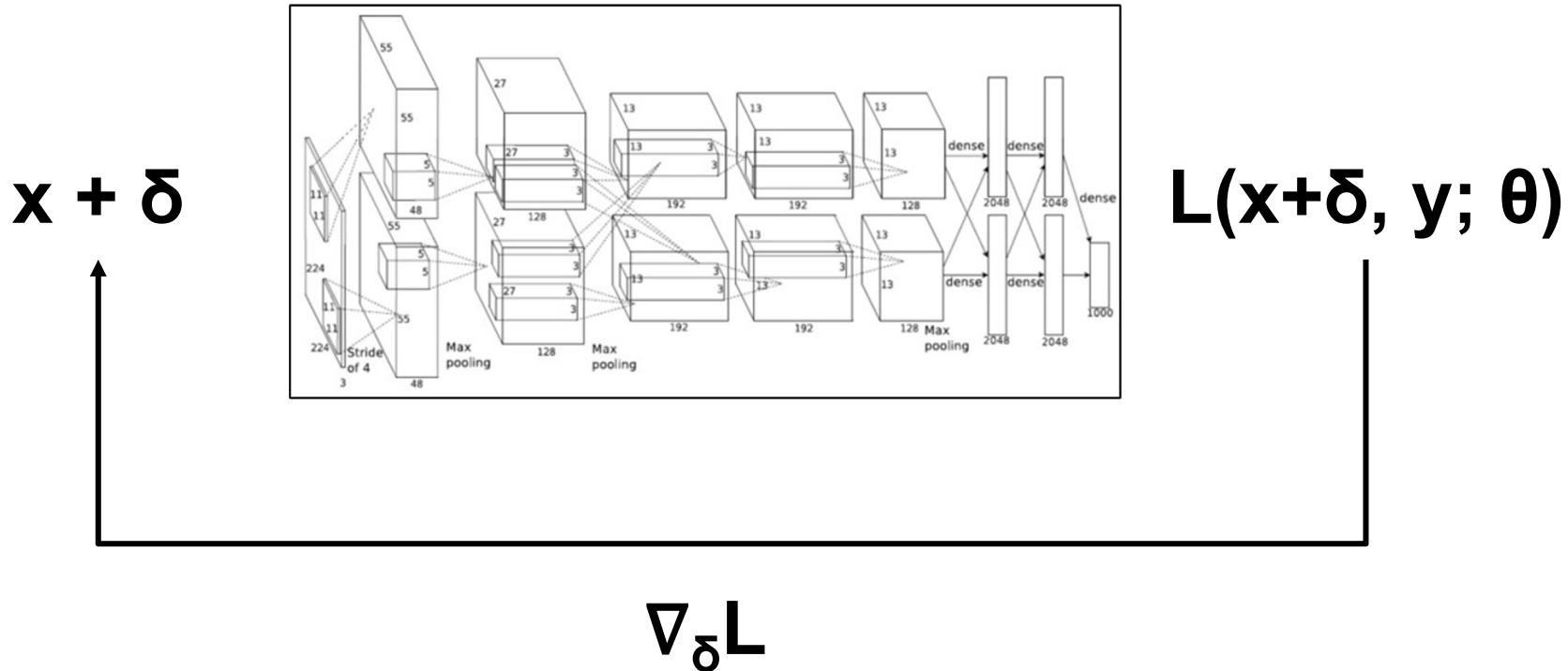
Perturbation is constrained to not exceed a certain budget, that is the *strength of the attack*

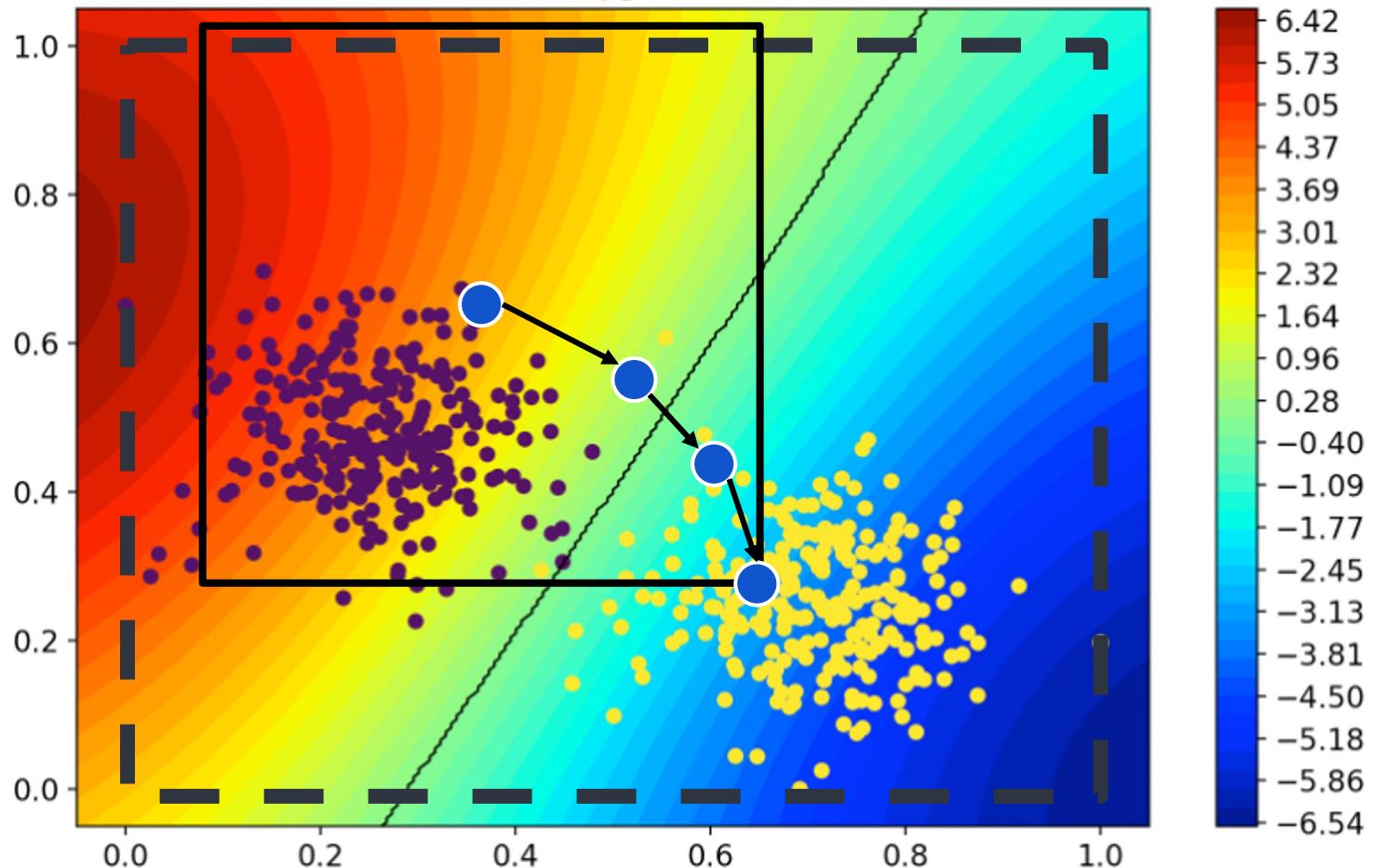
## Input-domain constraints

Bounds that force the attack to create valid samples inside the input space

**Keep this in mind, we will be back on constraints when addressing malware**

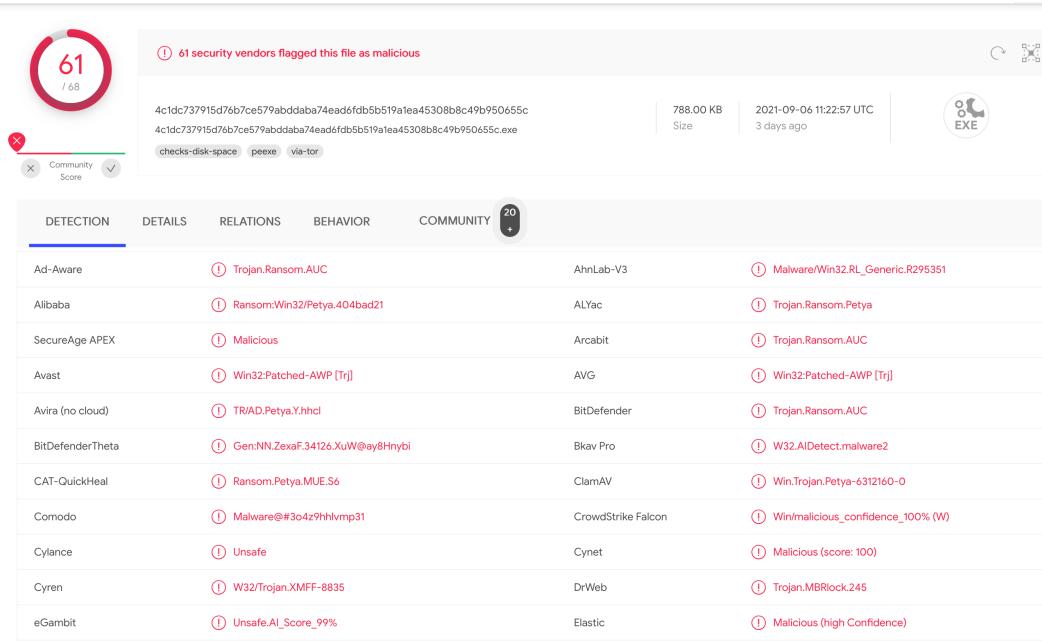
# Solutions with Gradients





[Biggio, Roli, *Wild Patterns*, Pattern Recognition 2018 <https://arxiv.org/abs/1712.03141>]  
[Pintor et al. *Indicators of Attack Failures*, NeurIPS 2022, <https://arxiv.org/pdf/2106.09947.pdf>]

# Reality-check check on Evaluations



The screenshot shows a detailed analysis of a file flagged as malicious by 61 security vendors. The interface includes a summary bar at the top with a 'Community Score' of 61/68, a file hash, size (788.00 KB), timestamp (2021-09-06 11:22:57 UTC), and a 'check-disk-space' button. Below this is a table of detections from various vendors:

Detection	Description	Vendor	Details
Ad-Aware	① Trojan.Ransom.AUC	AhnLab-V3	① Malware/Win32.RL_Generic.R295351
Alibaba	① Ransom:Win32/Petya.404bad21	ALYac	① Trojan.Ransom.Petya
SecureAge APEX	① Malicious	Arcabit	① Trojan.Ransom.AUC
Avast	① Win32:Patched-AWP [Trj]	AVG	① Win32:Patched-AWP [Trj]
Avira (no cloud)	① TR/AD.Petya.Y.hndl	BitDefender	① Trojan.Ransom.AUC
BitDefenderTheta	① Gen:NN.ZexaF.34126.XuW@ay8Hnybi	Bkav Pro	① W32.AIDetect.malware2
CAT-QuickHeal	① Ransom.Petya.MUE.S6	ClamAV	① Win.Trojan.Petya-6312160-0
Comodo	① Malware@#3o4z9hhlvmp31	CrowdStrike Falcon	① Win/malicious_confidence_100% (W)
Cylance	① Unsafe	Cynet	① Malicious (score: 100)
Cyren	① W32/Trojan.XMFF-8835	DrWeb	① Trojan.MBRlock.245
eGambit	① Unsafe.AI_Score_99%	Elastic	① Malicious (high Confidence)

Most models are hosted on private servers

Detection performed in cloud

No gradients can be computed!

# Solutions without Gradients

**Not always possible to compute gradients**

Model under test might be non-differentiable, or it might not be available but it can only be queried.

*Two ways then:*

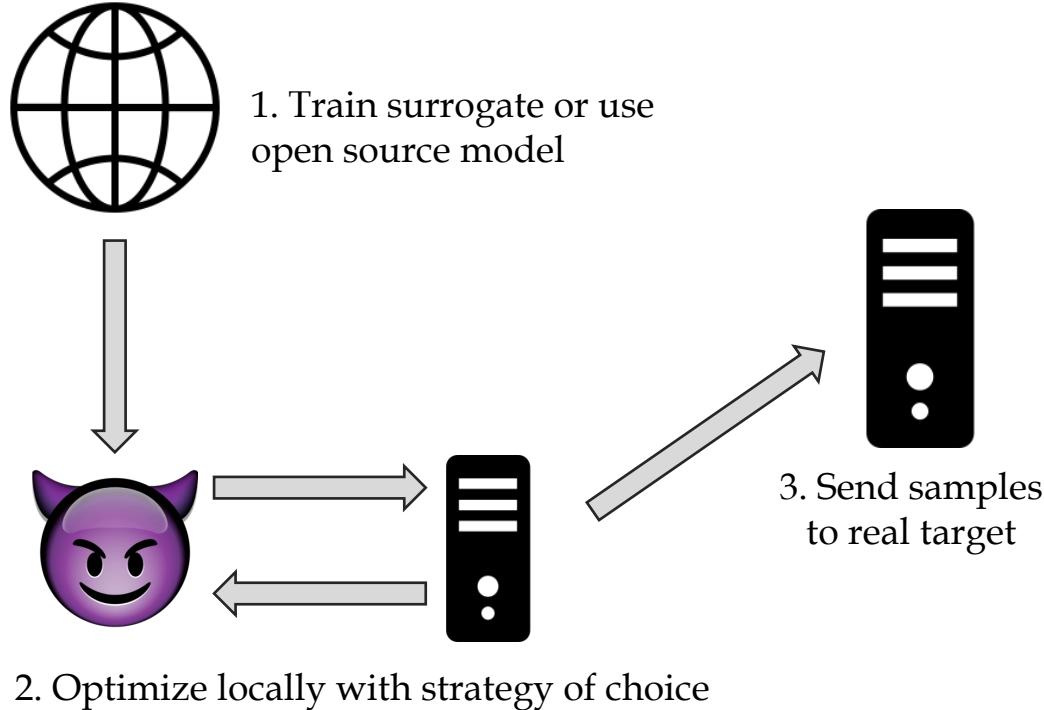
## Transfer Evaluations

Compute attacks on owned or similar model and test results on real target

## Query-based Optimization

Use the answers of the target to modify the parameters of noise

# Transfer Evaluations



## Craft surrogate classifier

Deploy or construct locally a model that behave similarly to the one we want to test

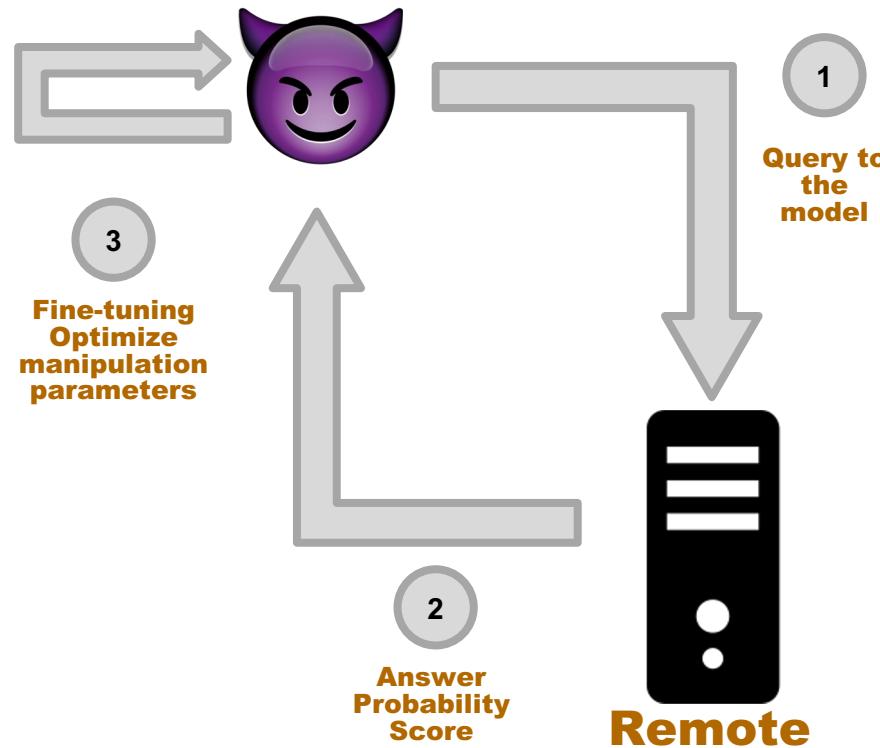
## Use strong attack

Attack the surrogate with the best attack in the arsenal

## Finger-cross, and test target

Send the computed attacks to the target, and hope they work as intended

# Query-based Optimization of Attacks



## Query the target

Ignoring its internals, send sample and wait for answer

## Update parameters of manipulation

Fine-tune manipulations by comparing responses

## Byte-per-byte optimization

Optimizer find best patterns, query after query

(more details later in the course)

# Digression: LLM Prompt Injections?



Plenty of hype around  
vulnerabilities of LLMs

BUT

These are EXACTLY  
optimization techniques applied  
on machine learning models with  
textual input

No more, no less

# Hands-on exercises time

<https://github.com/pralab/secml>

## SecML: Secure and Explainable Machine Learning in Python

status alpha python 3.6 | 3.7 | 3.8 platform linux | macos | windows license Apache-2.0

SecML is an open-source Python library for the **security evaluation** of Machine Learning algorithms. It is equipped with **evasion** and **poisoning** adversarial machine learning attacks, and it can **wrap models and attacks** from other different frameworks.

```
from secml.adv.attacks import CAttackEvasionPGD

distance = 'l2' # type of perturbation 'l1' or 'l2'
dmax = 2.5 # maximum perturbation
lb, ub = 0., 1. # bounds of the attack space. None for unbounded
y_target = None # None if untargeted, specify target label otherwise

# Should be chosen depending on the optimization problem
solver_params = {
    'eta': 0.5, # step size of the attack
    'max_iter': 100, # number of gradient descent steps
}

attack = CAttackEvasionPGD(classifier=secml_model,
                           distance=distance,
                           dmax=dmax,
                           solver_params=solver_params,
                           y_target=y_target)

adv_pred, scores, adv_ds, f_obj = attack.run(x, y)
```

Jupyter notebooks to understand how to apply simple adversarial attacks against example classifiers

## Lab 4:

# Hands-on tutorial on SecML Library

[aka.ms/malware-lab4](http://aka.ms/malware-lab4)

## Lab 5:

# Simple evasion attacks against machine learning models

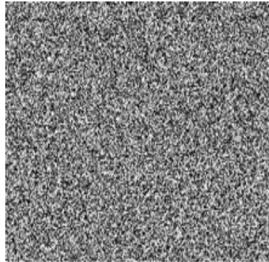
[aka.ms/malware-lab5](http://aka.ms/malware-lab5)

# Adversarial EXEmple from evil pixels to malicious bytes

# We can apply same attacks! Right?



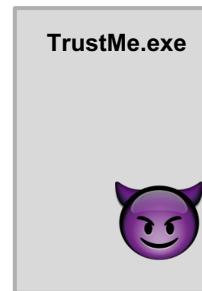
toucan (97%)



adversarial noise

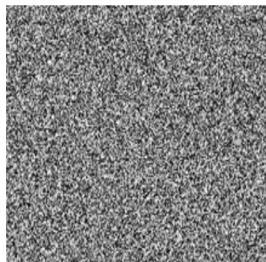


cat (95%)



TrustMe.exe

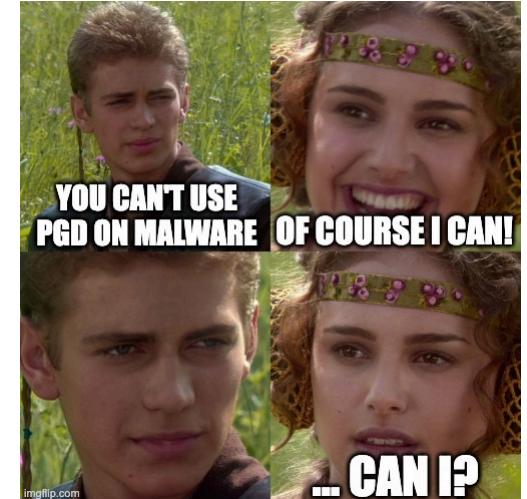
malware (98%)



adversarial noise



Not runnable anymore!



# Problem 1: no distance function!

Programs and images are encoded in bytes

RGB is “continuous”, code instructions are not!

Distance between programs is **undefined**

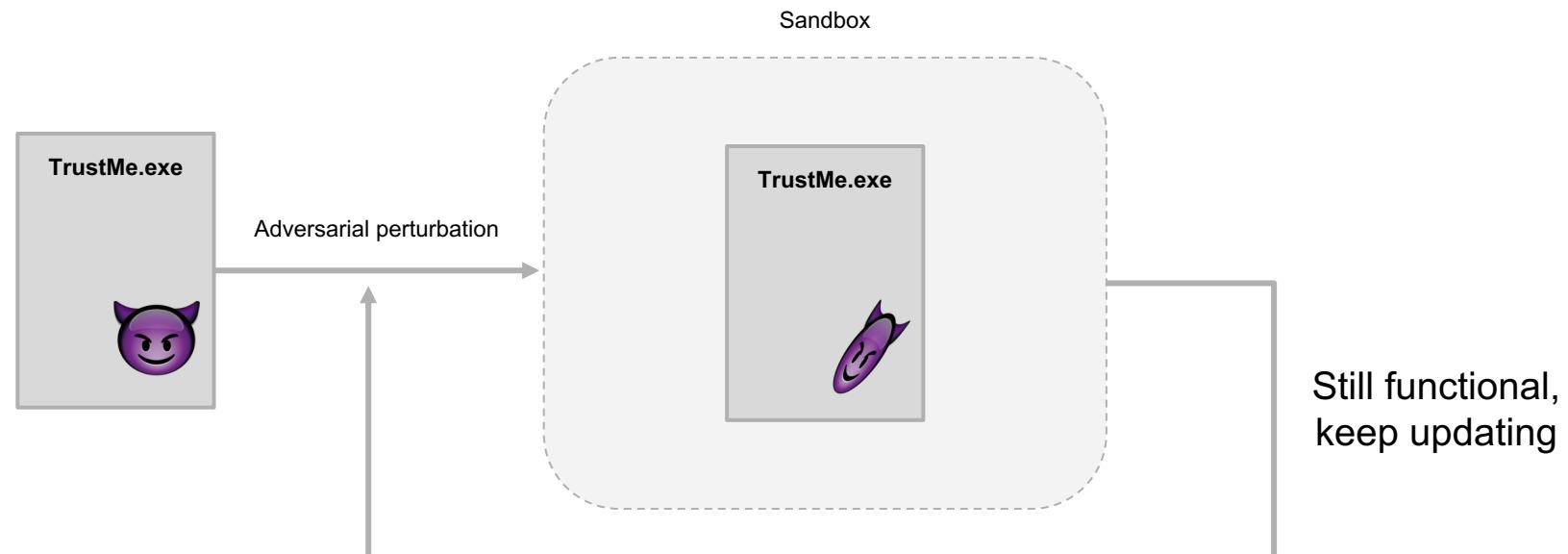
Example: ASCII table

What does  $\sqrt{a' - b'}$  means?

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	
0	0 000	NUL (null)	32 20 040	&#32; ;	Space	64 40 100	&#64; ;	Ø	96 60 140	&#96; ;	`	96 60 140	&#96; ;	Ø	96 60 140	&#96; ;	Ø	96 60 140	&#96; ;
1	1 001	SOH (start of heading)	33 21 041	&#33; !	;	65 41 101	&#65; ;	A	97 61 141	&#97; ;	a	97 61 141	&#97; ;	a	97 61 141	&#97; ;	a	97 61 141	&#97; ;
2	2 002	STX (start of text)	34 22 042	&#34; "	"	66 42 102	&#66; ;	B	98 62 142	&#98; ;	b	98 62 142	&#98; ;	b	98 62 142	&#98; ;	b	98 62 142	&#98; ;
3	3 003	ETX (end of text)	35 23 043	&#35; #	#	67 43 103	&#67; ;	C	99 63 143	&#99; ;	c	99 63 143	&#99; ;	c	99 63 143	&#99; ;	c	99 63 143	&#99; ;
4	4 004	EOT (end of transmission)	36 24 044	&#36; ¢	¢	68 44 104	&#68; ;	D	100 64 144	&#100; ;	d	100 64 144	&#100; ;	d	100 64 144	&#100; ;	d	100 64 144	&#100; ;
5	5 005	ENQ (enquiry)	37 25 045	&#37; %	%	69 45 105	&#69; ;	E	101 65 145	&#101; ;	e	101 65 145	&#101; ;	e	101 65 145	&#101; ;	e	101 65 145	&#101; ;
6	6 006	ACK (acknowledge)	38 26 046	&#38; &	&	70 46 106	&#70; ;	F	102 66 146	&#102; ;	f	102 66 146	&#102; ;	f	102 66 146	&#102; ;	f	102 66 146	&#102; ;
7	7 007	BEL (bell)	39 27 047	&#39; `	`	71 47 107	&#71; ;	G	103 67 147	&#103; ;	g	103 67 147	&#103; ;	g	103 67 147	&#103; ;	g	103 67 147	&#103; ;
8	8 010	BS (backspace)	40 28 050	&#40; (	(	72 48 110	&#72; ;	H	104 68 150	&#104; ;	h	104 68 150	&#104; ;	h	104 68 150	&#104; ;	h	104 68 150	&#104; ;
9	9 011	TAB (horizontal tab)	41 29 051	&#41; )	)	73 49 111	&#73; ;	I	105 69 151	&#105; ;	i	105 69 151	&#105; ;	i	105 69 151	&#105; ;	i	105 69 151	&#105; ;
10	A 012	LF (NL line feed, new line)	42 2A 052	&#42; *	*	74 4A 112	&#74; ;	J	106 6A 152	&#106; ;	j	106 6A 152	&#106; ;	j	106 6A 152	&#106; ;	j	106 6A 152	&#106; ;
11	B 013	VT (vertical tab)	43 2B 053	&#43; +	+	75 4B 113	&#75; ;	K	107 6B 153	&#107; ;	k	107 6B 153	&#107; ;	k	107 6B 153	&#107; ;	k	107 6B 153	&#107; ;
12	C 014	FF (NP form feed, new page)	44 2C 054	&#44; ,	,	76 4C 114	&#76; ;	L	108 6C 154	&#108; ;	l	108 6C 154	&#108; ;	l	108 6C 154	&#108; ;	l	108 6C 154	&#108; ;
13	D 015	CR (carriage return)	45 2D 055	&#45; -	-	77 4D 115	&#77; ;	M	109 6D 155	&#109; ;	m	109 6D 155	&#109; ;	m	109 6D 155	&#109; ;	m	109 6D 155	&#109; ;
14	E 016	SO (shift out)	46 2E 056	&#46; .	.	78 4E 116	&#78; ;	N	110 6E 156	&#110; ;	n	110 6E 156	&#110; ;	n	110 6E 156	&#110; ;	n	110 6E 156	&#110; ;
15	F 017	SI (shift in)	47 2F 057	&#47; /	/	79 4F 117	&#79; ;	O	111 6F 157	&#111; ;	o	111 6F 157	&#111; ;	o	111 6F 157	&#111; ;	o	111 6F 157	&#111; ;
16	10 020	DLE (data link escape)	48 30 060	&#48; 0	0	80 50 120	&#80; ;	P	112 70 160	&#112; ;	p	112 70 160	&#112; ;	p	112 70 160	&#112; ;	p	112 70 160	&#112; ;
17	11 021	DC1 (device control 1)	49 31 061	&#49; 1	1	81 51 121	&#81; ;	Q	113 71 161	&#113; ;	q	113 71 161	&#113; ;	q	113 71 161	&#113; ;	q	113 71 161	&#113; ;
18	12 022	DC2 (device control 2)	50 32 062	&#50; 2	2	82 52 122	&#82; ;	R	114 72 162	&#114; ;	r	114 72 162	&#114; ;	r	114 72 162	&#114; ;	r	114 72 162	&#114; ;
19	13 023	DC3 (device control 3)	51 33 063	&#51; 3	3	83 53 123	&#83; ;	S	115 73 163	&#115; ;	s	115 73 163	&#115; ;	s	115 73 163	&#115; ;	s	115 73 163	&#115; ;
20	14 024	DC4 (device control 4)	52 34 064	&#52; 4	4	84 54 124	&#84; ;	T	116 74 164	&#116; ;	t	116 74 164	&#116; ;	t	116 74 164	&#116; ;	t	116 74 164	&#116; ;
21	15 025	NAK (negative acknowledge)	53 35 065	&#53; 5	5	85 55 125	&#85; ;	U	117 75 165	&#117; ;	u	117 75 165	&#117; ;	u	117 75 165	&#117; ;	u	117 75 165	&#117; ;
22	16 026	SYN (synchronous idle)	54 36 066	&#54; 6	6	86 56 126	&#86; ;	V	118 76 166	&#118; ;	v	118 76 166	&#118; ;	v	118 76 166	&#118; ;	v	118 76 166	&#118; ;
23	17 027	ETB (end of trans. block)	55 37 067	&#55; 7	7	87 57 127	&#87; ;	W	119 77 167	&#119; ;	w	119 77 167	&#119; ;	w	119 77 167	&#119; ;	w	119 77 167	&#119; ;
24	18 030	CAN (cancel)	56 38 070	&#56; 8	8	88 58 130	&#88; ;	X	120 78 170	&#120; ;	x	120 78 170	&#120; ;	x	120 78 170	&#120; ;	x	120 78 170	&#120; ;
25	19 031	EM (end of medium)	57 39 071	&#57; 9	9	89 59 131	&#89; ;	Y	121 79 171	&#121; ;	y	121 79 171	&#121; ;	y	121 79 171	&#121; ;	y	121 79 171	&#121; ;
26	1A 032	SUB (substitute)	58 3A 072	&#58; :	:	90 5A 132	&#90; ;	Z	122 7A 172	&#122; ;	z	122 7A 172	&#122; ;	z	122 7A 172	&#122; ;	z	122 7A 172	&#122; ;
27	1B 033	ESC (escape)	59 3B 073	&#59; ;	;	91 5B 133	&#91; [	L	123 7B 173	&#123; {	{	123 7B 173	&#123; {	{	123 7B 173	&#123; {	{	123 7B 173	&#123; {
28	1C 034	FS (file separator)	60 3C 074	&#60; <	<	92 5C 134	&#92; \	I	124 7C 174	&#124;		124 7C 174	&#124;		124 7C 174	&#124;		124 7C 174	&#124;
29	1D 035	GS (group separator)	61 3D 075	&#61; =	=	93 5D 135	&#93; ]	J	125 7D 175	&#125; }	}	125 7D 175	&#125; }	}	125 7D 175	&#125; }	}	125 7D 175	&#125; }
30	1E 036	RS (record separator)	62 3E 076	&#62; >	>	94 5E 136	&#94; ^	~	126 7E 176	&#126; ~	~	126 7E 176	&#126; ~	~	126 7E 176	&#126; ~	~	126 7E 176	&#126; ~
31	1F 037	US (unit separator)	63 3F 077	&#63; ?	?	95 5F 137	&#95; _	DEL	127 7F 177	&#127; DEL	DEL	127 7F 177	&#127; DEL	DEL	127 7F 177	&#127; DEL	DEL	127 7F 177	&#127; DEL

Source: [www.asciitable.com](http://www.asciitable.com)

# Problem 2: functionality, is that you?



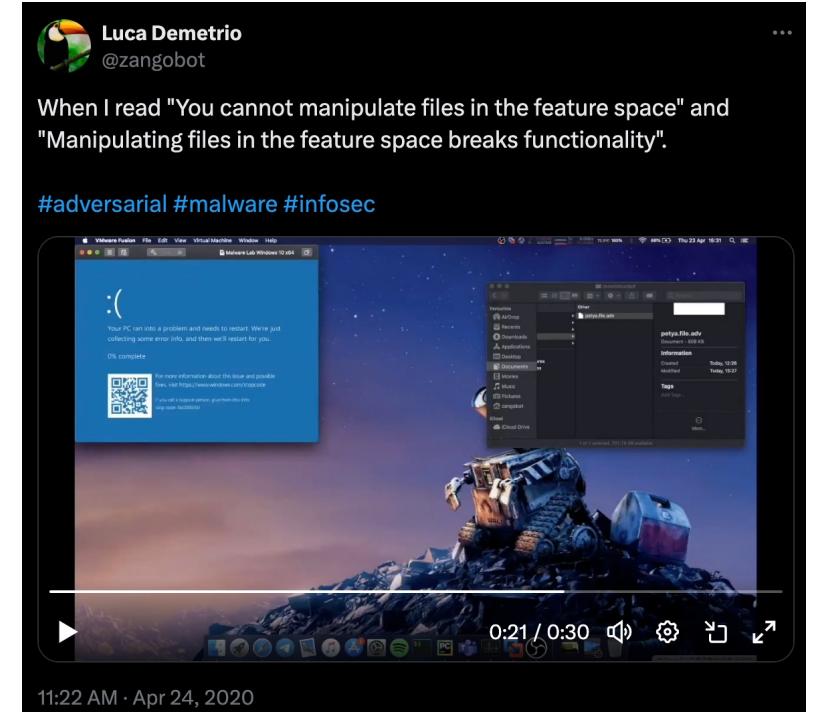
Song et al., *MAB-Malware: A Reinforcement Learning Framework for Attacking Static Malware Classifiers*, AsiaCCS 2022  
Castro et al., *AIMED: Evolving Malware with Genetic Programming to Evasion Detection*, TrustCom/BigDataSE 2019

# Discouraging research?

In 2019 ~ 2020, people were telling us that the problem too hard to solve

ALSO ME IN 2019 ~ 2020

<https://x.com/zangobot/status/1253615316569120768>



# Adversarial EXEmpleS: change of paradigm

The problem IS possible to solve,  
thanks to some twists:

1. **Formulate** the minimization problem differently
2. **Study the format** that represent programs, and understand how to exploit its ambiguities
3. **Create new algorithms** that inject or replace content

# Different formulation

$$\min_{\delta} L(x + \delta, y; \theta)$$

**Network architecture in the loss**  
All the internals of a  
neural network / shallow model are  
hidden inside the loss

**Additive Manipulation**  
Input samples are injected with  
additive noise, without any concern  
on the structure of the file

# Different formulation

$$\min_{\delta} L(f(\phi(h(x; \delta)), y))$$



## Model function and features

Need to explicit the model function and the features, since they might be non differentiable

## Practical Manipulations

No additions, but a complex function that handles format specification by design

Demetrio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

# Different formulation

$$\min_{\delta} L(f(\phi(h(x; \delta)), y))$$

## Define the Optimizer

Depending on the differentiability of the components, pick a gradient-based or gradient-free algorithm

## Define the Manipulations

Study the format, understand its ambiguities, and write manipulations that do not break the original functionality

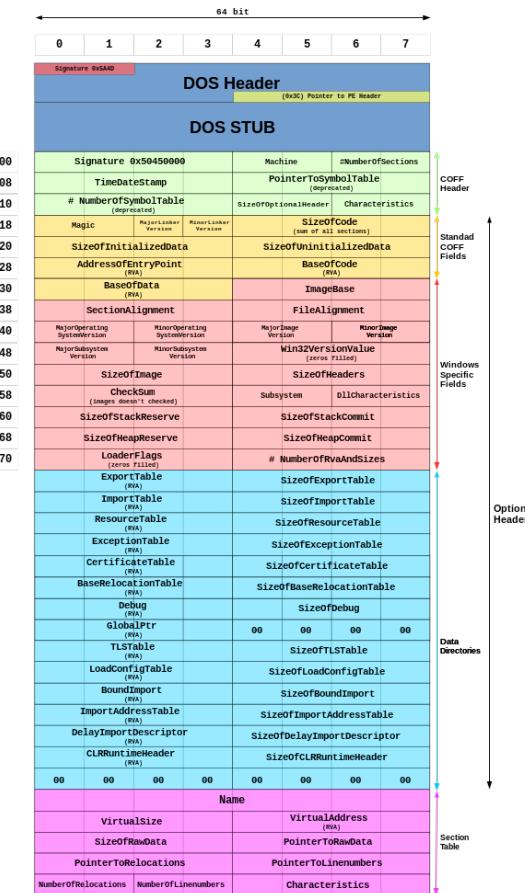
Demetrio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

# Manipulations of Windows PE File Format

# Recap: PE format

Format adapted for “modern” programs (from Windows NT 3.1 on)

Before there were other formats, one is the DOS (kept for retrocompatibility)

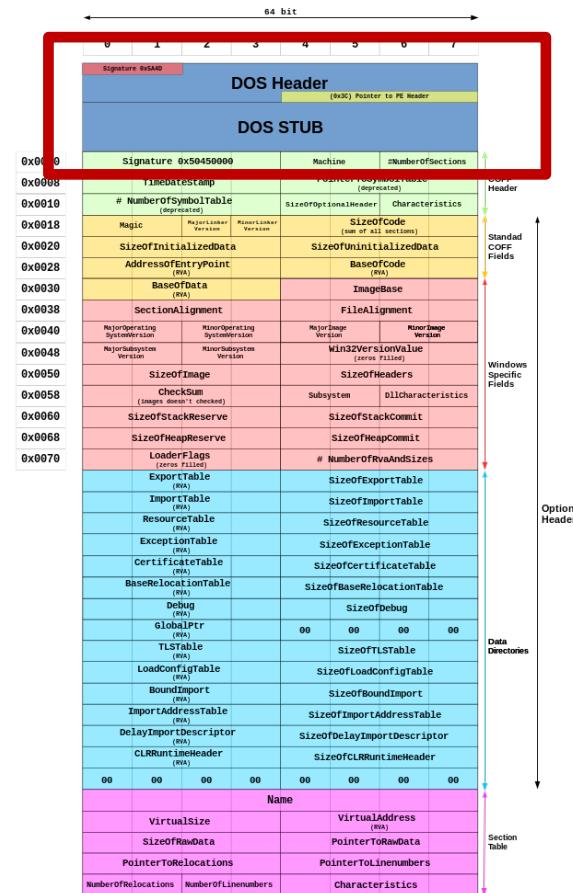


# Recap: PE format

## DOS Header + Stub

Metadata for DOS program

Executing a modern program in DOS will trigger the “This program cannot be run in DOS mode” output

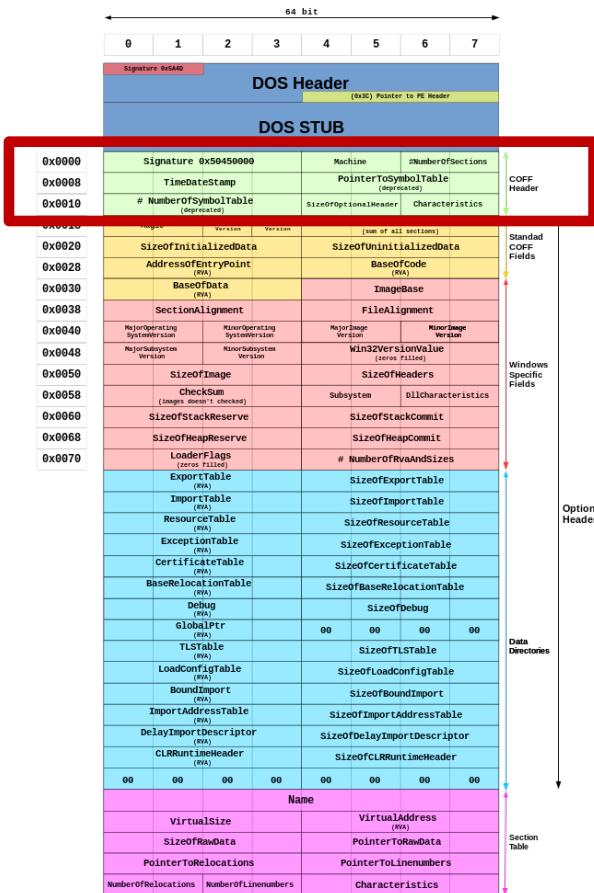


# Recap: PE format

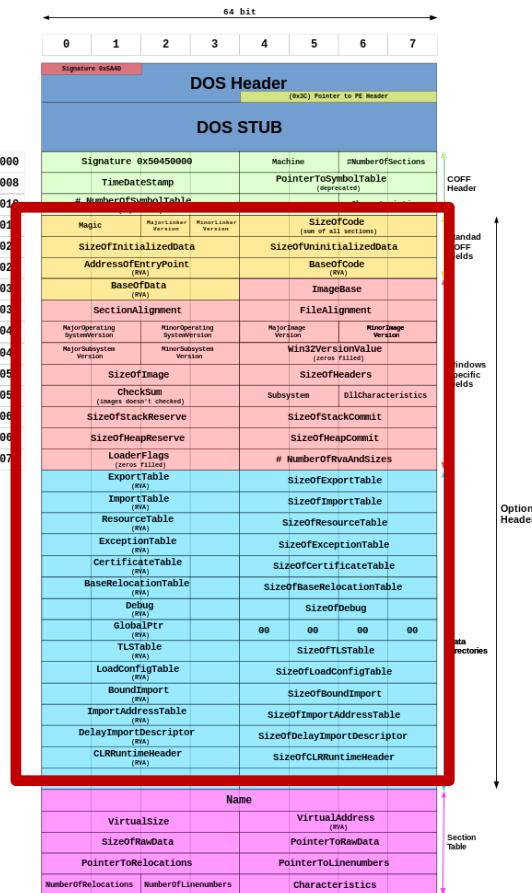
## PE Header

Real metadata of the program

Describes general information of the file



# Recap: PE format



## Optional Header

Spoiler: not optional at all :)

Instructs the loader where to find each object inside the file

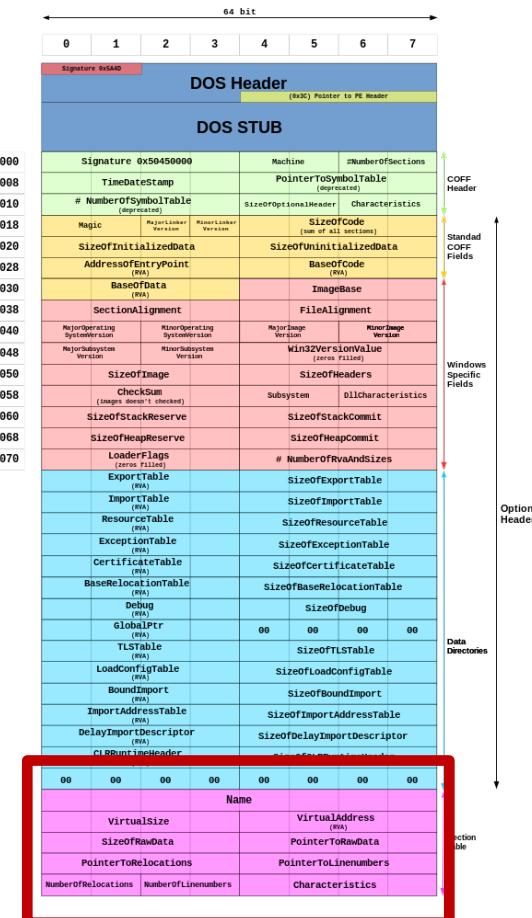
# Recap: PE format

## Section Table and Sections

Describes where to find code, initialized data, resources, etc to the loader

These are “sections”, and each has a “section entry” with its characteristics

Examples: code is “.text”, read-only data is “.rodata”, resources are “.rsc”, and counting



# How programs are loaded

## 1 Headers

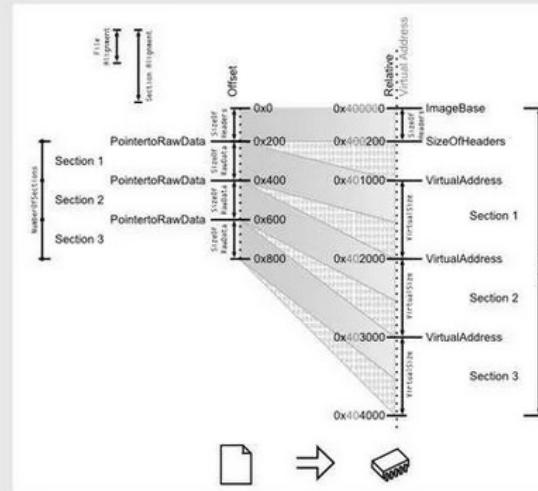
the DOS Header is parsed  
the PE Header is parsed  
(its offset is DOS Header's e\_lfanew)  
the Optional Header is parsed  
(it follows the PE Header)

## 2 Sections table

Sections table is parsed  
(it is located at: offset (OptionalHeader) + SizeOfOptionalHeader)  
it contains NumberOfSections elements  
it is checked for validity with alignments:  
*FileAlignments* and *SectionAlignments*

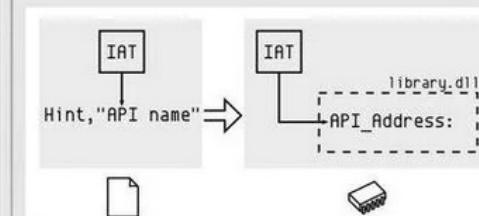
## 3 Mapping

the file is mapped in memory according to:  
the *ImageBase*  
the *SizeOfHeaders*  
the Sections table



## 4 Imports

DataDirectories are parsed  
they follow the *OptionalHeader*  
their number is *NumOfRVAAAndSizes*  
imports are always #2  
Imports are parsed  
each descriptor specifies a *DLLname*  
this DLL is loaded in memory  
*IAT* and *INT* are parsed simultaneously  
for each API in *INT*  
its address is written in the *IAT* entry



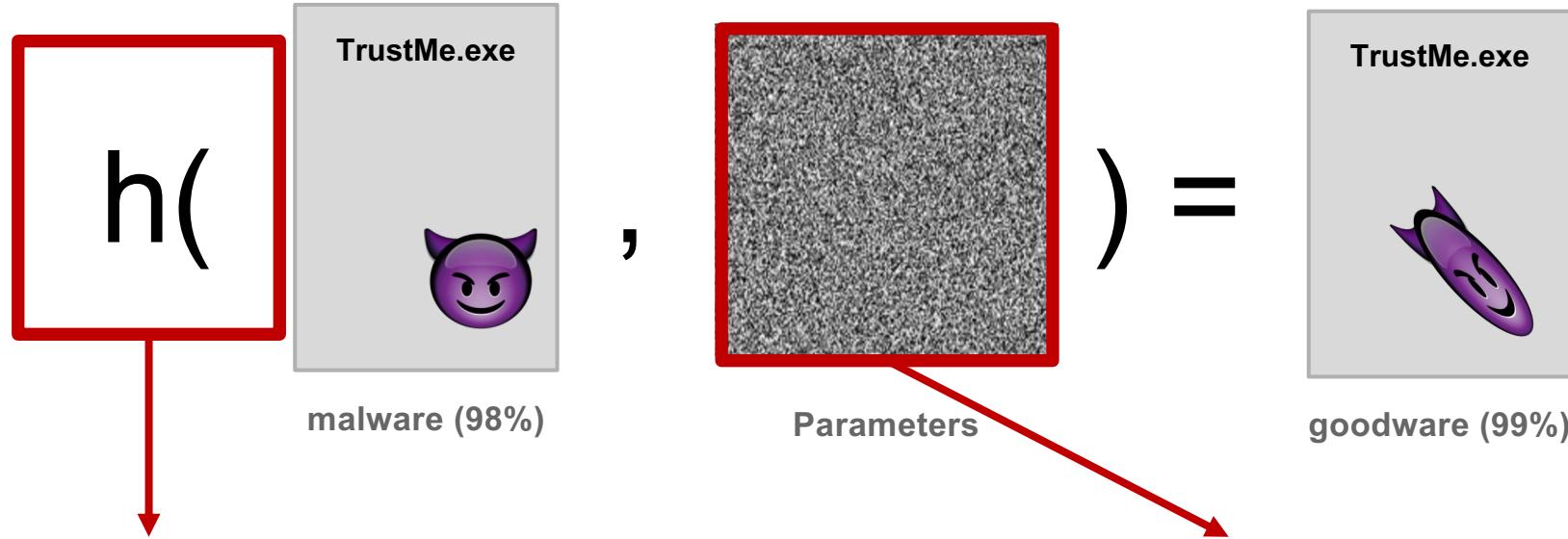
## 5 Execution

Code is called at the *EntryPoint*  
the calls of the code go via the *IAT* to the APIs



<https://code.google.com/archive/p/corkami/wikis/PE101.wiki>

# Practical Manipulations



## Practical Manipulation function

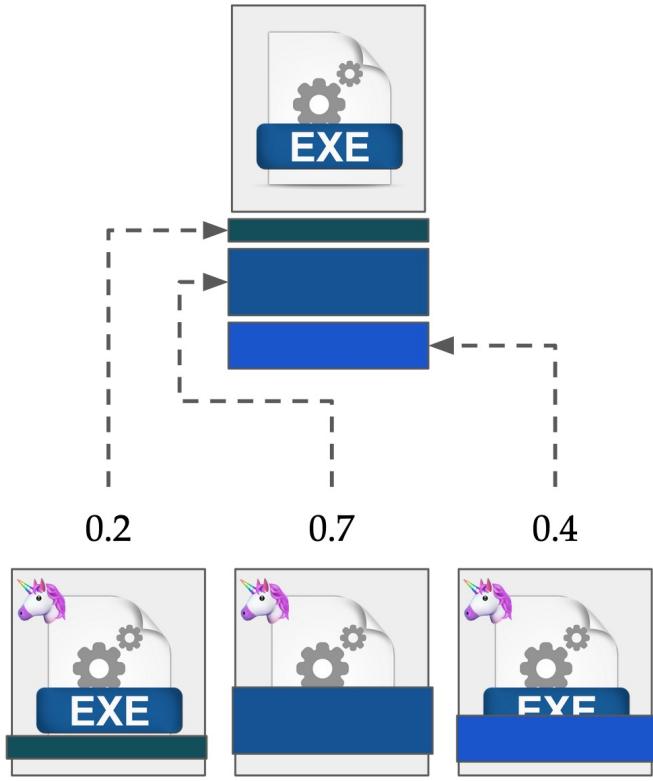
Alter file representation  
without destroying the structure  
and the functionalities and avoid  
usage of sandboxes

## Parameters

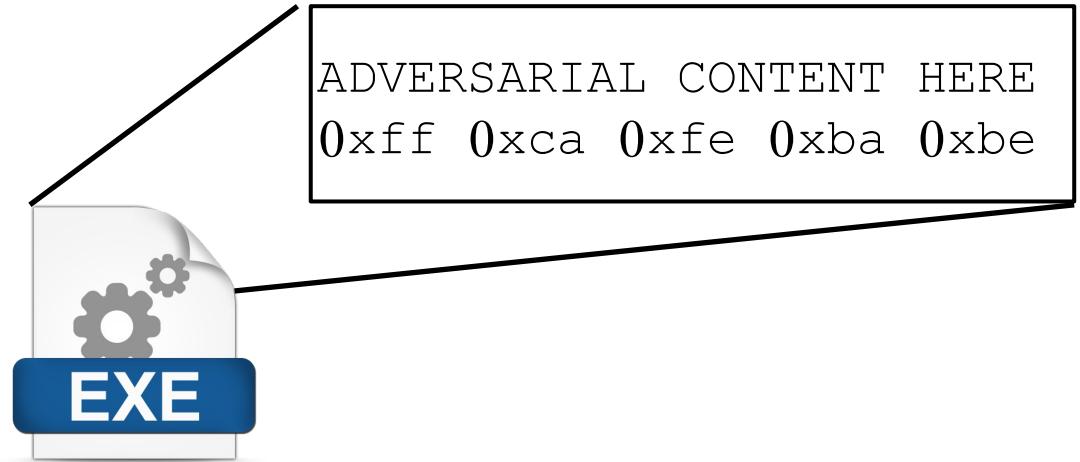
Manipulations are parametrized  
so an optimization algorithm can  
fine tune them

Demetrio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

# Structural Manipulations

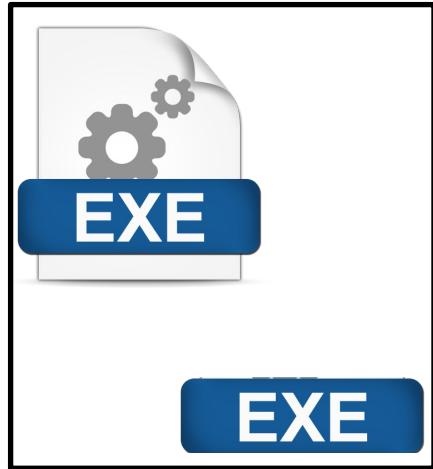


**Injecting content**  
Alter file structure to include  
more byte sequences



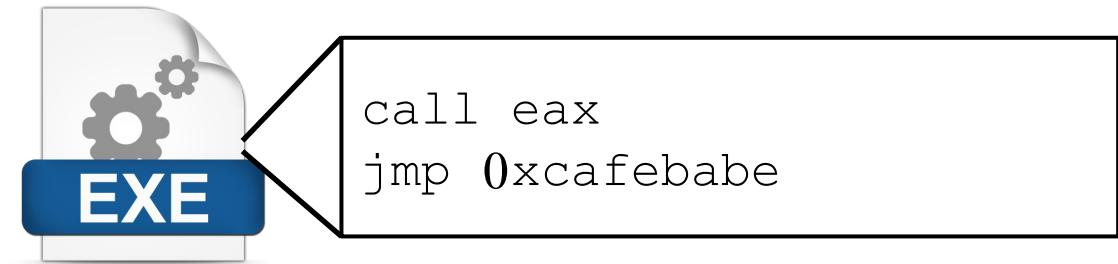
**Replacing content**  
Leverage ambiguous file format specifications  
to alter bytes that are not considered at runtime

# Behavioral Manipulations



## Packing and obfuscation

Encrypt program inside another one, or complicate the sequence of instructions



## Inject new execution flows

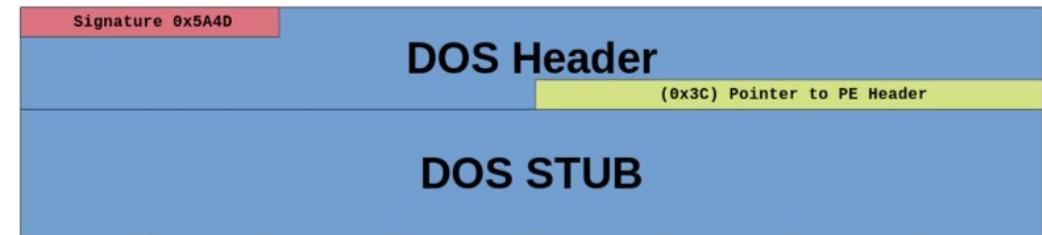
Call APIs, add loops, jump to new code sections, and more

# DOS Header Manipulation (1/2)

The attacker edit as many bytes as they want

**Untouched:** magic number MZ and offset to real PE header

Content loaded in memory, not executed



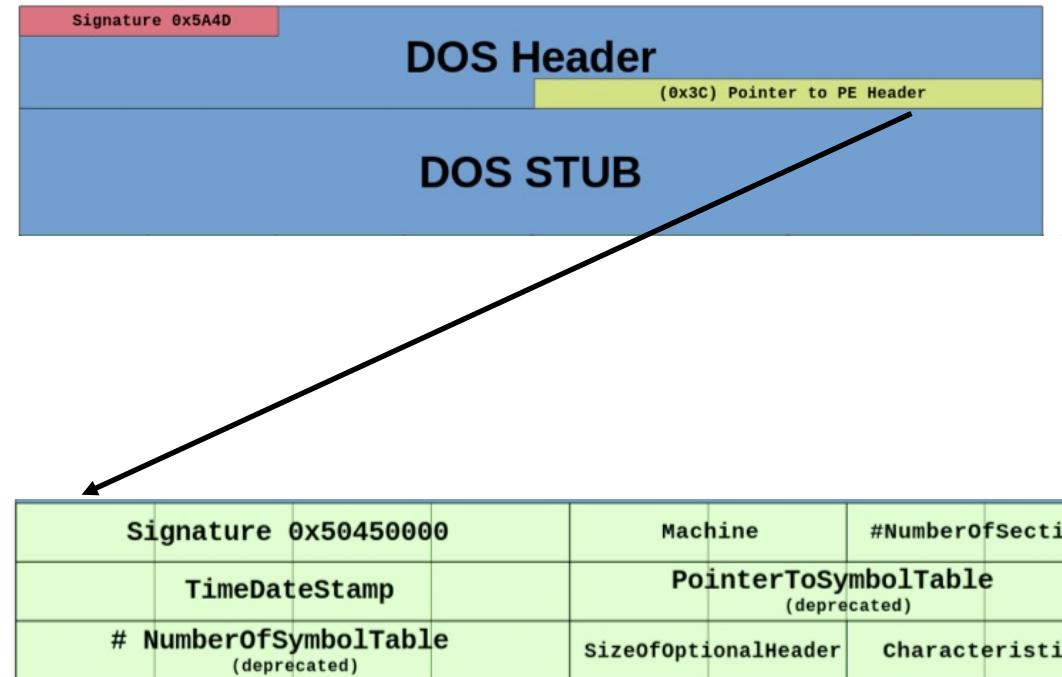
Demetrio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

# DOS Header Manipulation (2/2)

Exploit offset to real header, increment value

Insert arbitrary content between DOS header and PE header

Content loaded into memory, not executed



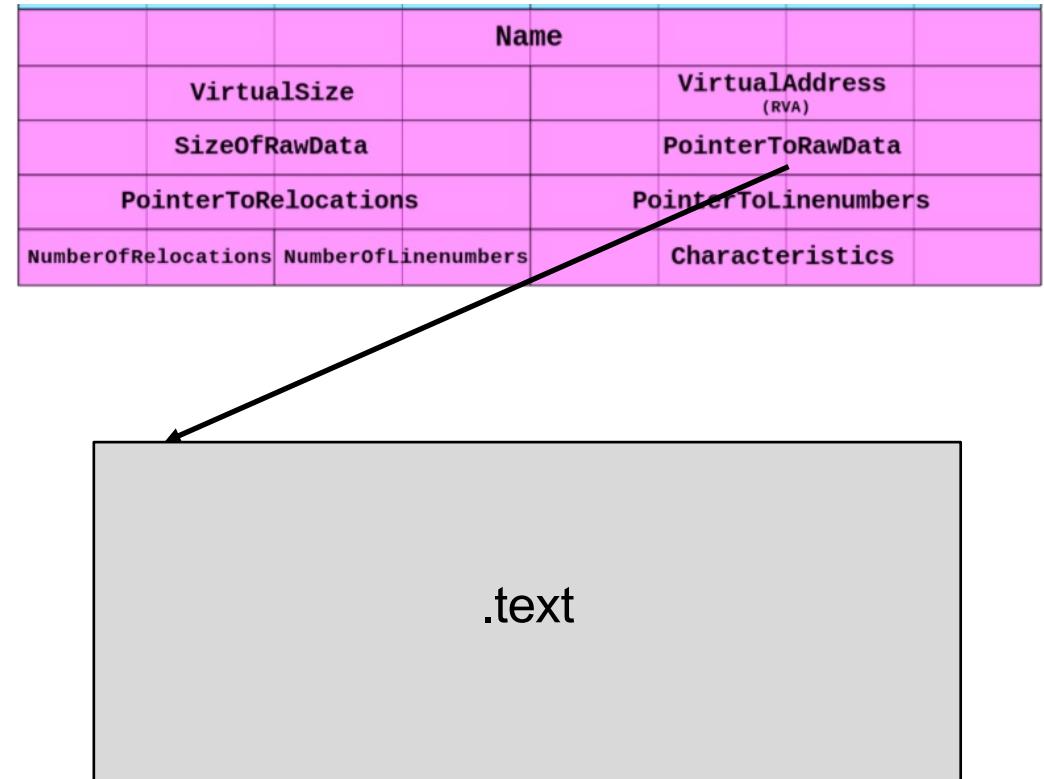
Demetrio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

# Content-shifting Manipulation

Exploit offset in section entry, increment to manipulate the loader in searching for section content

The attacker can inject content after the section table, or between sections

**NOT LOADED IN MEMORY**, skipped by the loader



Demetrio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

# Section-injection Manipulation

Manipulate section table to add new entry

Append chunk of bytes, referenced by newly added entry

Loaded in memory or not, depending by the characteristics set up inside the entry

Name			
	VirtualSize	VirtualAddress (RVA)	
	SizeOfRawData	PointerToRawData	
	PointerToRelocations	PointerToLinenumbers	
NumberOfRelocations	NumberOfLinenumbers	Characteristics	

Name			
	VirtualSize	VirtualAddress (RVA)	
	SizeOfRawData	PointerToRawData	
	PointerToRelocations	PointerToLinenumbers	
NumberOfRelocations	NumberOfLinenumbers	Characteristics	

.text

.adv

# Slack Manipulation

Section content is padded with 0 to keep file alignments

The attacker can rewrite such slack space

Loaded in memory, not executed



.text

.data

Demetrio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

# Padding Manipulation

Appending content at the end

Most trivial manipulation

Not loaded in memory



paint.exe

The diagram shows a file structure for 'paint.exe'. It consists of a light gray rectangular area representing the executable code and a dark green rectangular area representing padding or metadata at the bottom. The text 'paint.exe' is positioned to the right of the file structure.

Demetrio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

# Optimization Algorithms

# Choose your Fighter

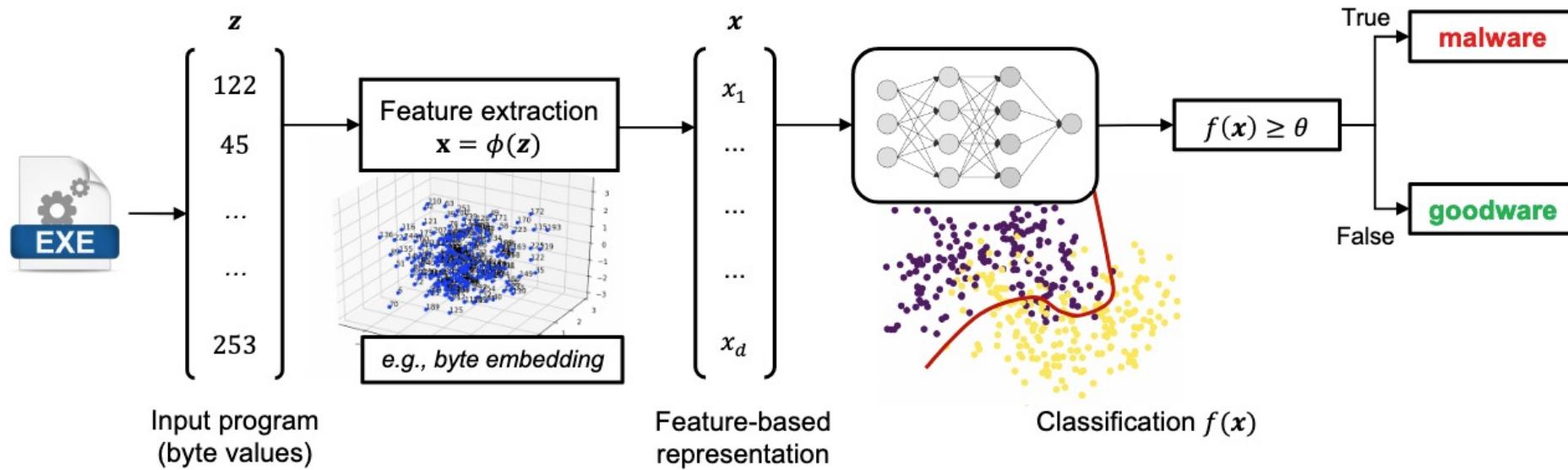
**Gradient-based**

Owning the model  
AND  
Model is differentiable

**Gradient-free**

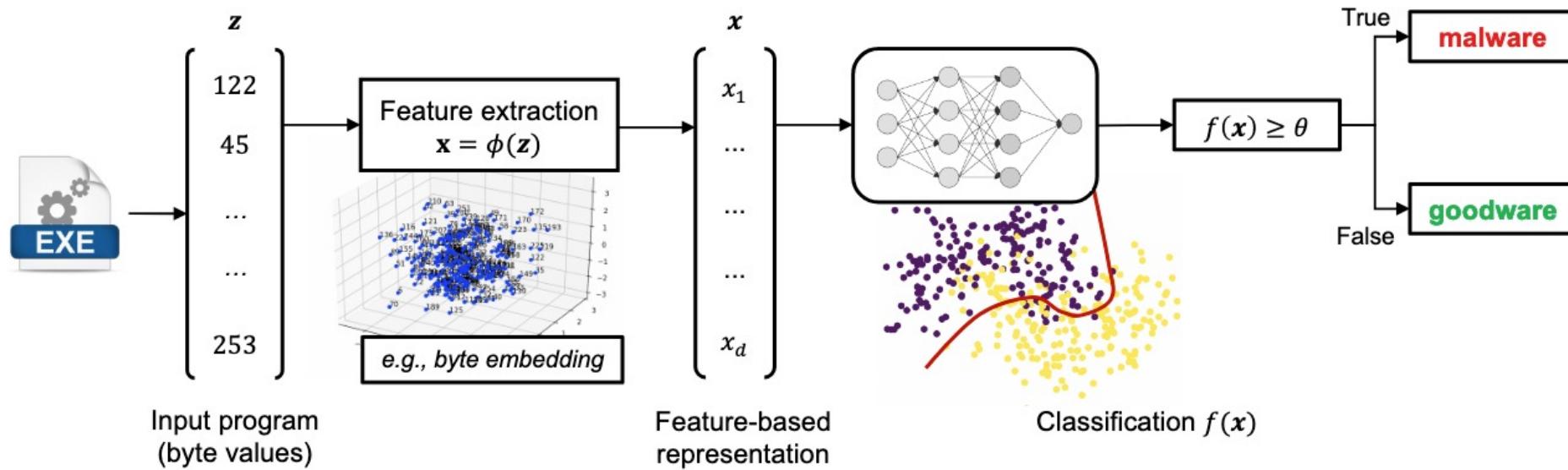
Model not accessible  
OR  
Model is not differentiable

# Gradient-based Attacks



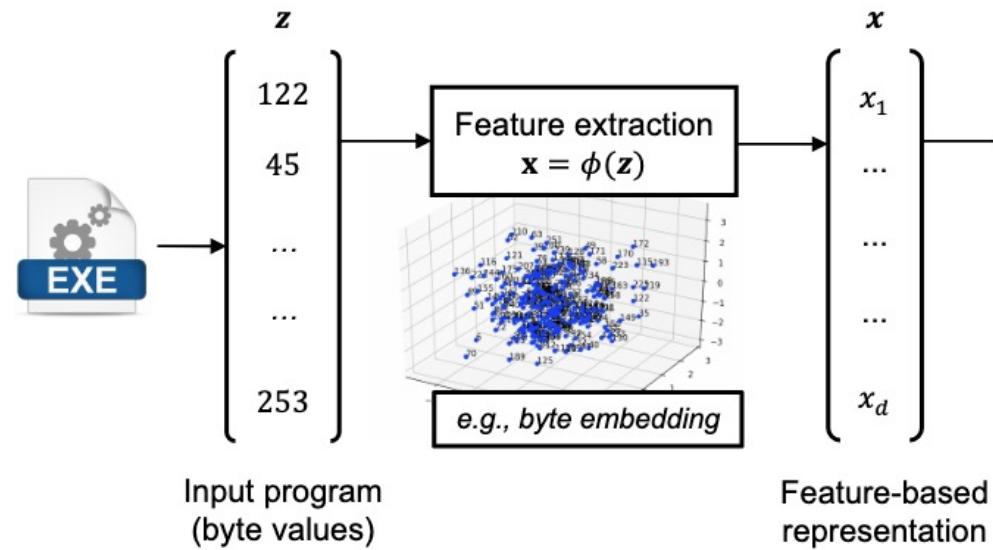
Models might be differentiable, so we can rely on gradient-descent strategies like images

# Gradient-based Attacks



Models might be differentiable, so we can rely on gradient descent strategies like images  
Bytes do not have a distance metric, a feature extractor is **ALWAYS** needed to compute something meaningful

# Embedding space again



All bytes are replaced with a vector learned at training time,  
where a distance metric is imposed...

... but the embedding layer is **not differentiable**

# Few differentiable operations!

$$\frac{\partial L}{\partial \delta} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial \phi} \frac{\partial \phi}{\partial h} \frac{\partial h}{\partial \delta}$$

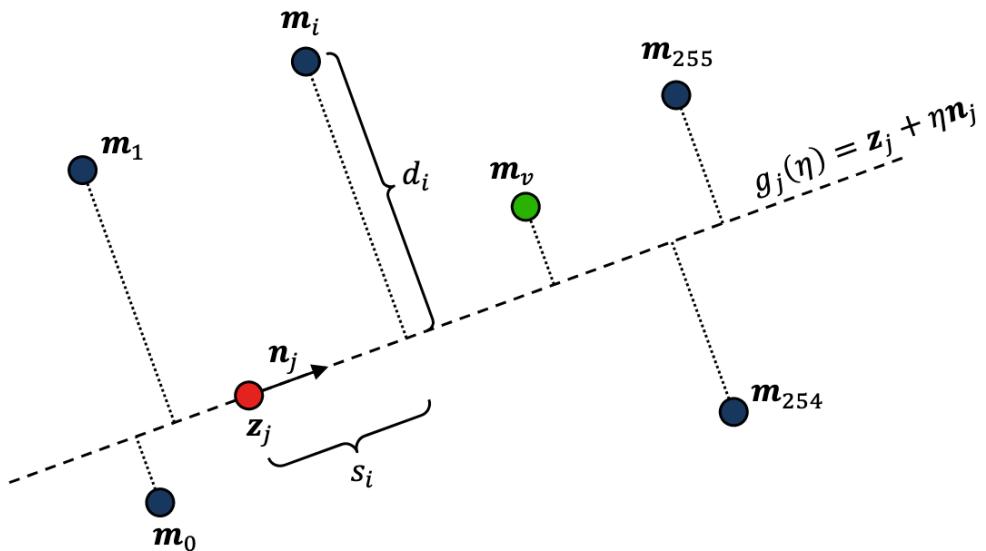


End-to-end gradient you  
would like to compute



**Non-differentiable  
manipulations and  
embedding!**

# Deal with discrete by changing optimization



Still gradient descent, but inside the **embedding space!**

Optimize where gradients are available and reconstruct bytes **after** the search

Demetrio, Biggio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021  
Kolosnjaji et al., *Adversarial malware binaries: Evading deep learning for malware detection in executables*, EUSIPCO 2018

# BGD: Byte Gradient Descent

```

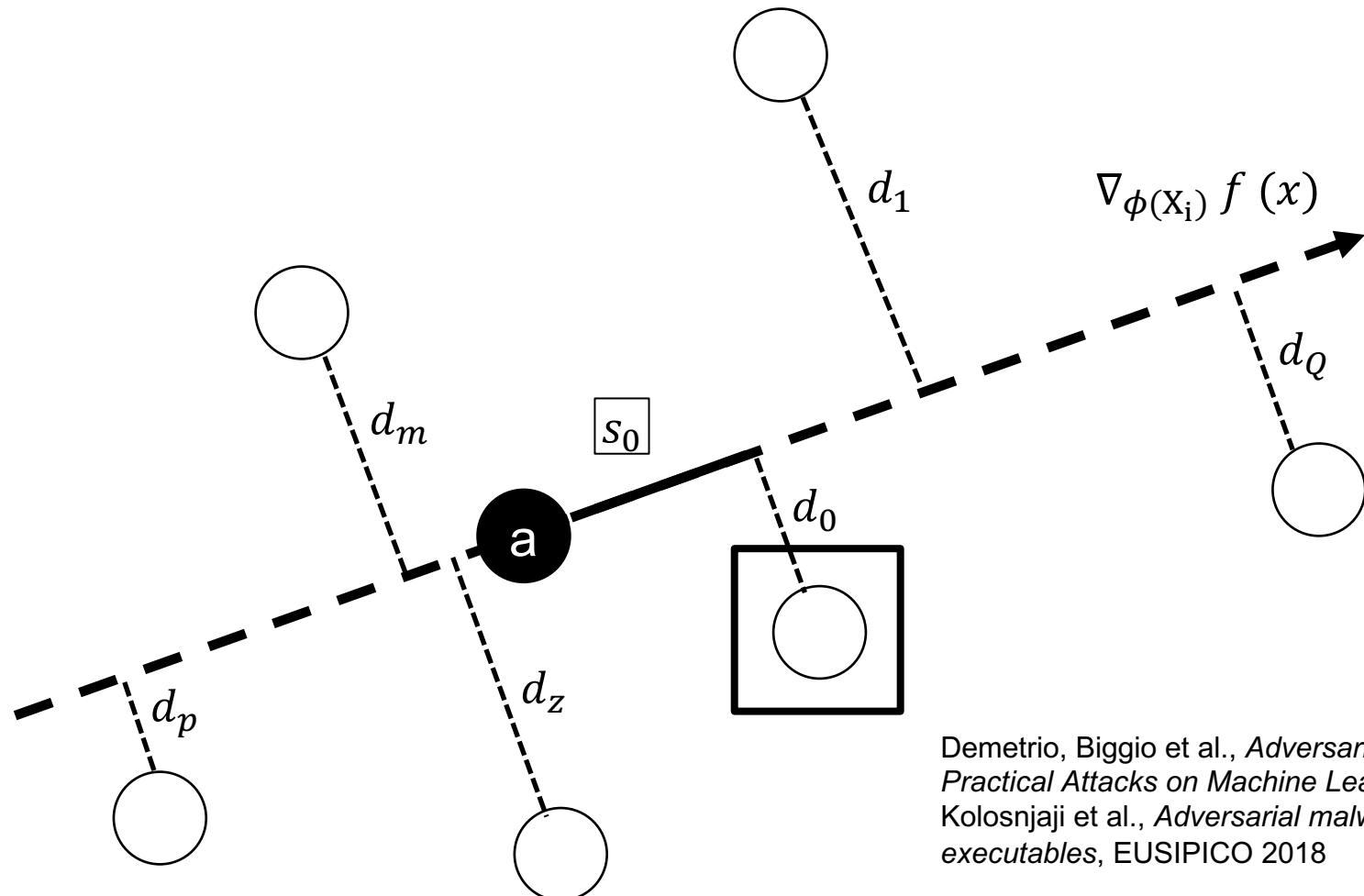
1  $E_i = \hat{\phi}(i), \forall i \in [0, 256]$ 
2  $t^{(0)} \in \mathcal{T}$ 
3 for  $i$  in  $[0, N - 1]$ 
4    $X \leftarrow \phi(h(z, t^{(i)}))$  // feat. space
5    $G \leftarrow -\nabla_X f(X) \odot m$ 
6    $g \leftarrow (\|G_0\|, \dots, \|G_n\|)$ 
7   for  $k$  in  $\text{argsort}(g)_{0, \dots, \gamma} \wedge g_k \neq 0$ 
8     for  $j$  in  $[0, \dots, 255]$ 
9        $S_{k,j} \leftarrow G_k^t \cdot (E_j - X_k)$ 
10       $\tilde{X}_{k,j} \leftarrow \|E_j - (X_k + G_k S_{k,j})\|_2$ 
11       $t_k^{(i+1)} \leftarrow \arg \min_{j: S_{k,j} > 0} \tilde{X}_{k,j}$  //input space
12     $t^\star \leftarrow t^{(N)}$ 
13   $z^\star \leftarrow h(z, t^\star)$ 
14 return  $z^\star$ 

```

1. Compute gradient in feature space
2. Define a way for replacing values  
For bytes: inverse look-up of embedding
3. Follow the direction of gradient and replace byte with other byte

Demetrio, Biggio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021  
Kolosnjaji et al., *Adversarial malware binaries: Evading deep learning for malware detection in executables*, EUSIPCO 2018

# BGD: Byte Gradient Descent



The process is repeated according to the **stepsize of the attack**, that quantifies how many bytes are modified at each iteration

Demetrio, Biggio et al., *Adversarial EXEmple: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021  
Kolosnjaji et al., *Adversarial malware binaries: Evading deep learning for malware detection in executables*, EUSIPCO 2018

# BGD: Byte Gradient Descent

At the end of each iteration, I  
need to replace one byte, **not an**  
**embedding value**

But each byte is chosen in the  
embedding space, reconstruction  
just invert the look-up function

$$\arg \min_{j:S_{k,j} > 0} \tilde{X}_{k,j}$$

# Choose your Fighter

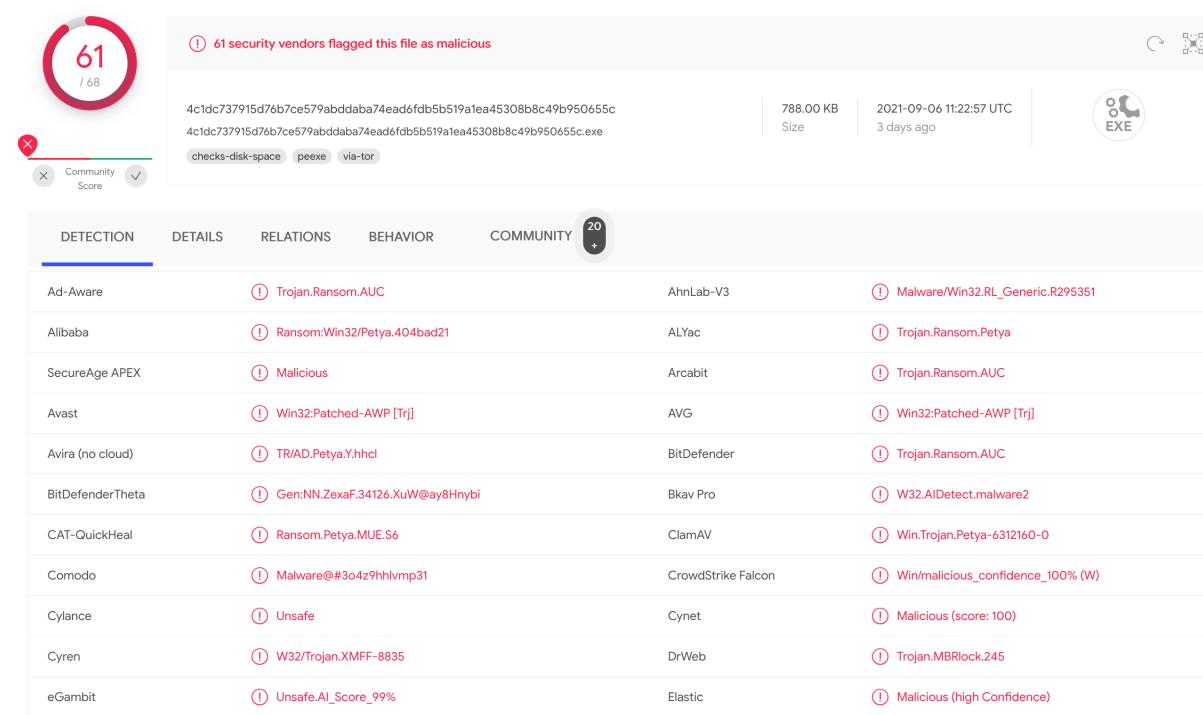
**Gradient-based**

Own the model  
AND  
Model is differentiable

**Gradient-free**

Model not accessible  
OR  
Model is not differentiable

# Reality check: most models are unavailable



The screenshot shows a digital interface for analyzing a file. At the top left, a circular progress bar indicates 61 out of 68 security vendors have flagged the file as malicious. The main area displays a table of vendor detections:

Vendor	Detection	Engine	Category
Ad-Aware	Trojan.Ransom.AUC	AhnLab-V3	Malware/Win32.RL_Generic.R295351
Alibaba	Ransom:Win32/Petya.404bad21	ALYac	Trojan.Ransom.Petya
SecureAge APEX	Malicious	Arcabit	Trojan.Ransom.AUC
Avast	Win32:Patched-AWP [Tr]	AVG	Win32:Patched-AWP [Tr]
Avira (no cloud)	TR/AD.Petya.Yhhcl	BitDefender	Trojan.Ransom.AUC
BitDefenderTheta	Gen>NN.ZexxF.34126.XuW@ay8Hnybi	Bkav Pro	W32.AIDetect.malware2
CAT-QuickHeal	Ransom.Petya.MUE.S6	ClamAV	Win.Trojan.Petya-6312160-0
Comodo	Malware@#3o4z9hhlvmp31	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cylance	Unsafe	Cynet	Malicious (score: 100)
Cyren	W32/Trojan.XMFF-8835	DrWeb	Trojan.MBRblock.245
eGambit	Unsafe.AI_Score_99%	Elastic	Malicious (high Confidence)

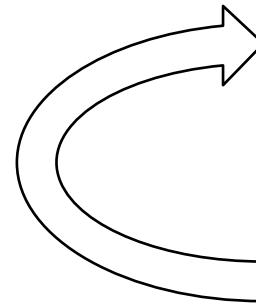
Most models are hosted on private servers

Detection performed in cloud

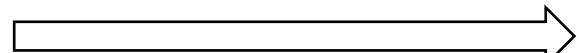
No gradients can be computed

# Query attacks

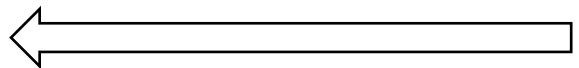
3. Perturb bytes of the sample, considering the scores from remote



1. Send sample to target



2. Obtain scores from remote



**Very slow if optimizer  
works byte-per-byte**

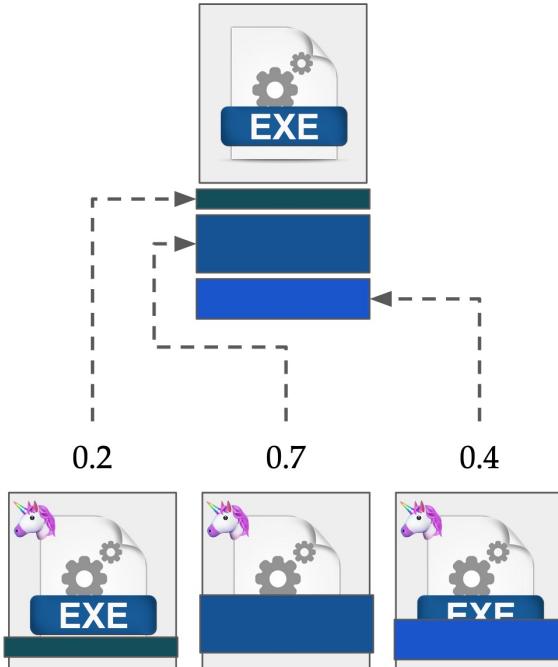
Demetrio et al., *Functionality-preserving Black-box Optimization of Adversarial Windows malware*, IEEE TIFS 2021

# GAMMA: Speeding up by injecting benign content

## Intuition

classifiers can be fooled by introducing content of the goodware class!

The optimizer explores less space, no modification byte-per-byte, but it relies on portions of goodware programs injected with practical manipulations



Demetrio et al., *Functionality-preserving Black-box Optimization of Adversarial Windows malware*, IEEE TIFS 2021

# (In)Famous example: CyLance

## Injecting bytes

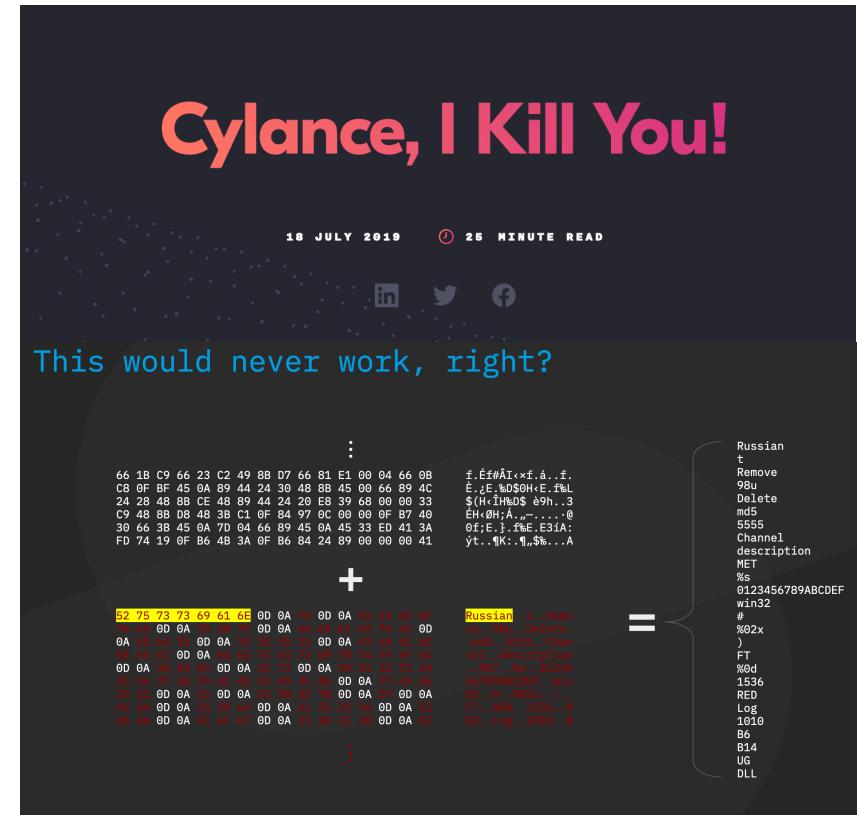
Reversing the code with some tricks, discovered that the model leverages STRINGS

## Inject “benign” values

Extract byte sequences from “Rocket League” and include them inside input executable

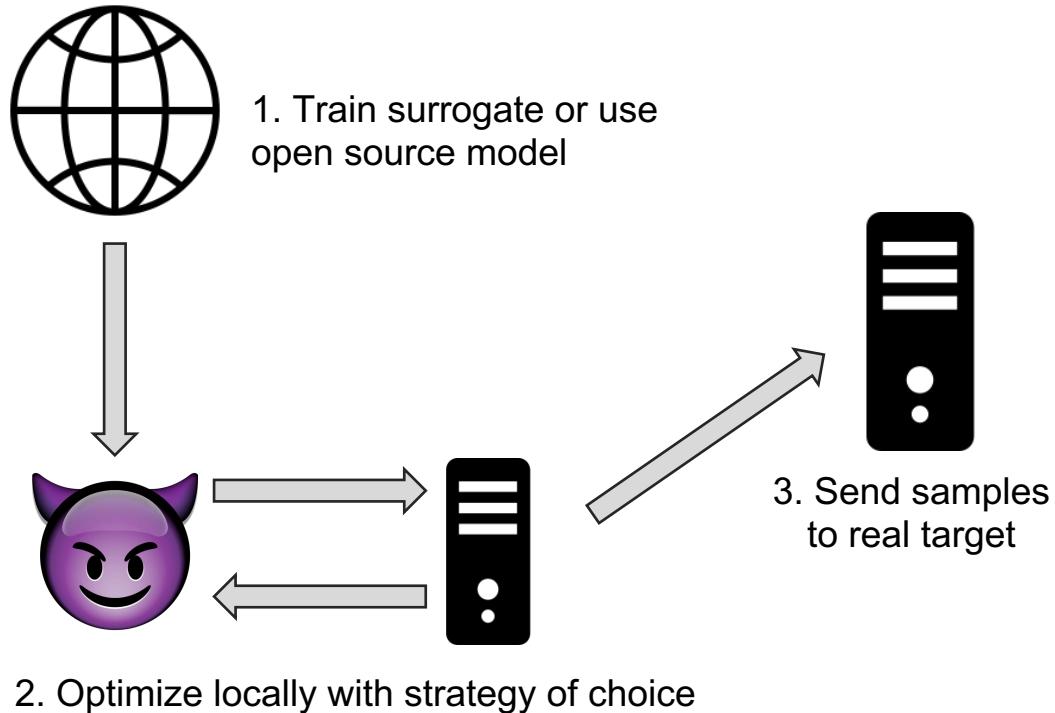
## Evasion completed!

The company rolled out an update to try to mitigate the issue



[Skylight Cyber. Cylance, I Kill You! <https://skylightcyber.com/2019/07/18/cylance-i-kill-you/>]

# Transfer Evaluations



**Choose a good malware detector as surrogate**  
As seen in Part 1, GBDT with EMBER is a good choice, since its predictive capabilities are far better than other models

# GAMMA against commercial products

<b>Malware</b>	<b>Random</b>	<b>Sect. Injection</b>
<b>AV1</b>	93.5%	85.5%
<b>AV2</b>	85.0%	78.0%
<b>AV3</b>	85.0%	46.0%
<b>AV4</b>	84.0%	83.5%
<b>AV5</b>	83.5%	79.0%
<b>AV6</b>	83.5%	82.5%
<b>AV7</b>	83.5%	54.5%
<b>AV8</b>	76.5%	71.5%
<b>AV9</b>	67.0%	54.5%
		16.5%

**Breaking signatures and patterns**  
GAMMA (transfer) reduces the performance of  
commercial products hosted on VirusTotal  
(and we'll touch upon this later)

Demetrio et al., *Functionality-preserving Black-box Optimization of Adversarial Windows malware*, IEEE TIFS 2021

## Lab 6:

# Crafting Adversarial EXEmplEs against different models

[aka.ms/malware-lab6](http://aka.ms/malware-lab6)

## Lab 7:

# Example of evasive C2C sample

[aka.ms/malware-lab7](http://aka.ms/malware-lab7)

## Lab 8:

# Effect of Adversarial EXEmplar against YARA rules

[aka.ms/malware-lab8](http://aka.ms/malware-lab8)

# Defending against Adversarial EXEmplEs

# Recap: Adversarial EXEmples

## Minimal byte perturbations

Many examples on how machine learning malware detectors can be bypassed with carefully-crafted input

## Ambiguities of file format

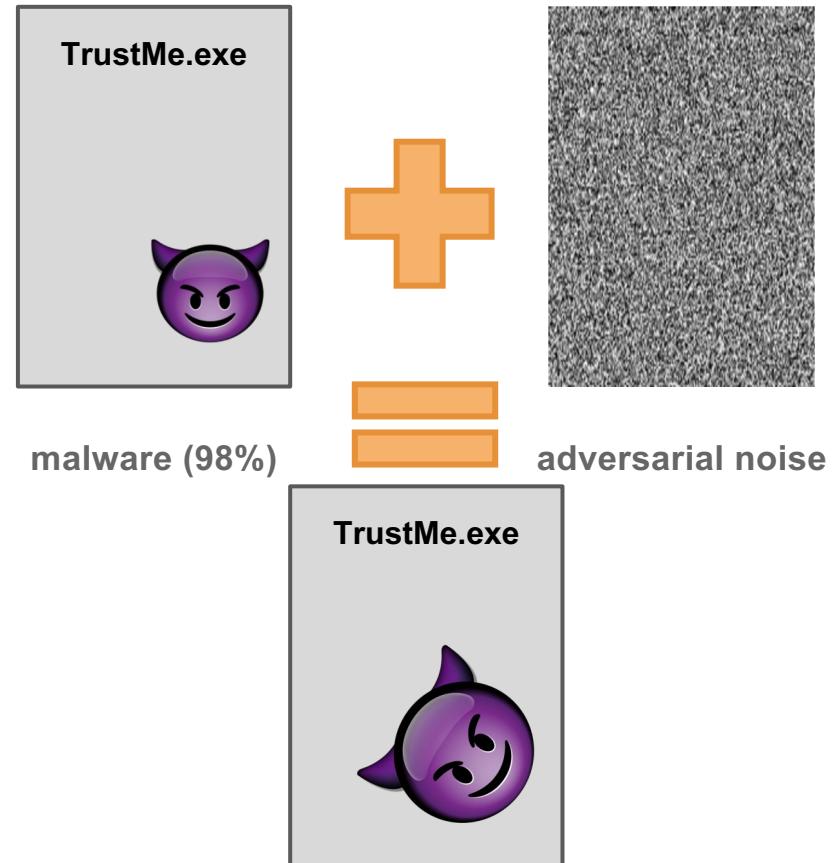
The Windows PE file format is redundant and many components are not used by the operating system at loading time, giving space to the attacker

## Math is unreliable

The domain is discrete, models work mostly with continuous values, and attackers can “fill the blanks” with adversarial manipulations inside this huge mathematical space

## How to avoid EXEmples?

Not clear how to patch this problem, but we isolated 4 relevant “claimed-to-be” robust malware detectors



## Adversarial EXEmple

Demetrio, Biggio, et al. "Adversarial EXEmples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection." TOPS 2021

# Heuristic defense

## Combination of pre-processing

Detect trivial manipulation, and then process input with ensemble of models

## Partially reproducible

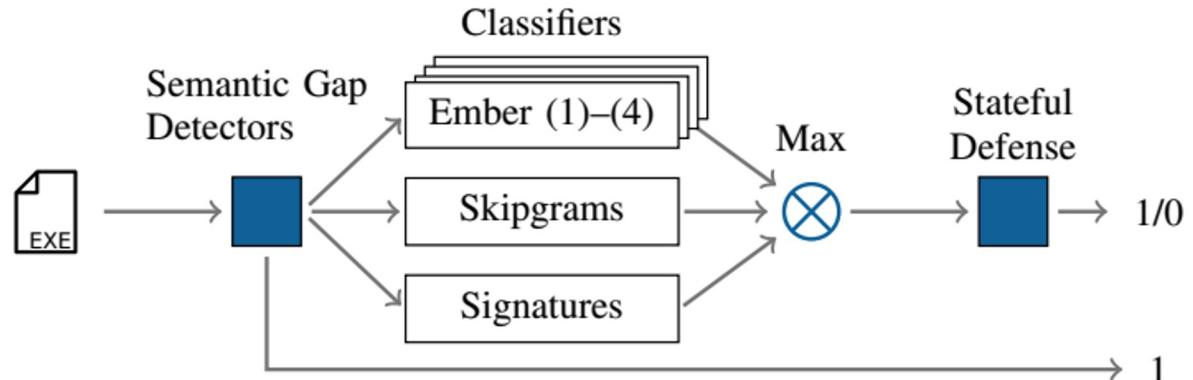
There are no pre-trained available, but code is available online

<https://github.com/EQuiw/2020-evasion-competition>

## Against All Odds: Winning the Defense Challenge in an Evasion Competition with Diversification

Erwin Quiring, Lukas Pirch, Michael Reimsbach, Daniel Arp, Konrad Rieck

Technische Universität Braunschweig  
Braunschweig, Germany



Quiring et al. *Against all the Odds: Winning the Defense Challenge in an Evasion Competition with Diversification*, preprint

# Adversarial Training

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[ \max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

## Train with EXEmplar

Computing state-of-the art attacks and include them inside the training set (process is repeated until the achievement of the desired robustness)

Madry et al. "Towards Deep Learning Models resistant to Adversarial Attacks" ICLR 2018

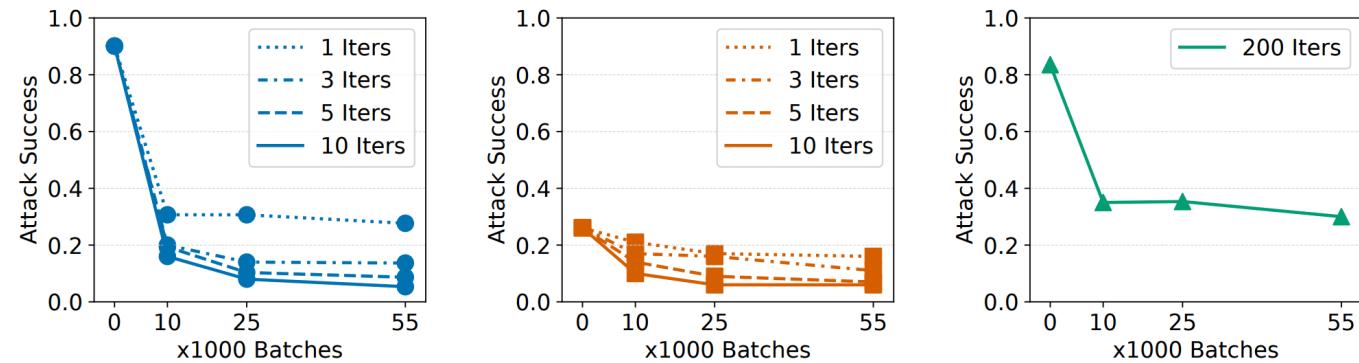
# Adversarial Training

## Alter code

Leverage behavioral manipulations to rewrite part of assembly code

## Published, not reproducible

There are no pre-trained available, nor public source code that can be used to train a model. The technique is known, but the attacks used for this paper as well are closed



**Adversarial Training for Raw-Binary Malware Classifiers**

Keane Lucas  
*Carnegie Mellon University*

Lujo Bauer  
*Carnegie Mellon University*

Samruddhi Pai  
*Carnegie Mellon University*

Michael K. Reiter  
*Duke University*

Weiran Lin  
*Carnegie Mellon University*

Mahmood Sharif  
*Tel Aviv University*

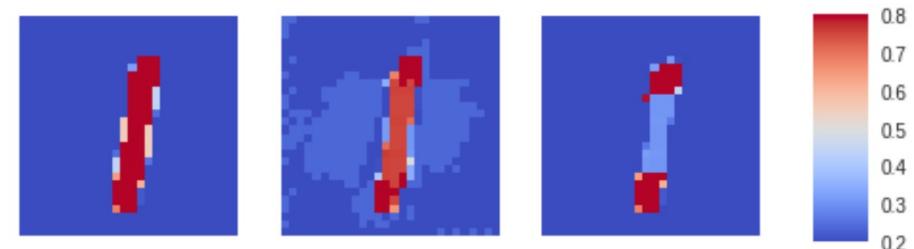
# Non-negative Networks

## Malicious contributions

Intuition: classification is based on addition of small malicious triggers, until a threshold is reached

## Remove negative weights

Train an end-to-end model, by clipping to positive values all the weights of the network



## Attacks constrained

On images, non-negative network force attacks to only tamper with meaningful information

# Non-negative Networks

## Pre-trained available

Testing through model trained on EMBER,  
released for a challenge  
(performances are debatable)

## Unpublished

Still a preprint, never published to either  
conferences, journals or workshops

## Hardly reproducible

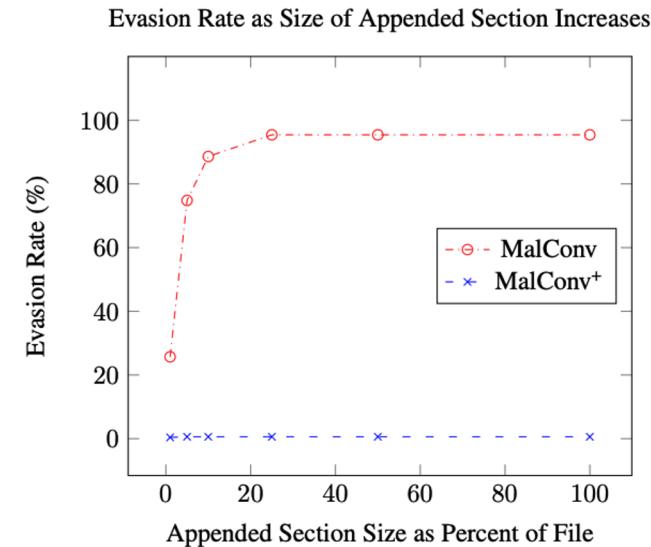
Even by trying to re-write code, many papers  
tried and failed to train NonNeg MalConv

## Non-Negative Networks Against Adversarial Attacks

William Fleshman,<sup>1</sup> Edward Raff,<sup>1,2</sup> Jared Sylvester,<sup>1,2</sup> Steven Forsyth,<sup>3</sup> Mark McLean<sup>1</sup>

<sup>1</sup>Laboratory for Physical Sciences, <sup>2</sup>Booz Allen Hamilton, <sup>3</sup>Nvidia

{william.fleshman, edraff, jared, mrmclea}@lps.umd.edu, sforsyth@nvidia.com



Fleshman et al. *Non-negative Networks Against Adversarial Attacks*, preprint

# Monotonic Classifiers

## Malicious contributions

Intuition: classification is based on addition of small malicious triggers, until a threshold is reached

## Gradient boosting decision tree

Use custom training process that trains an additive decision function

## Subset of features

Not using EMBER, but the authors propose a reduced features set that is harder to manipulate

## Not reproducible

There are no pre-trained available, nor public source code that can be used to train a model

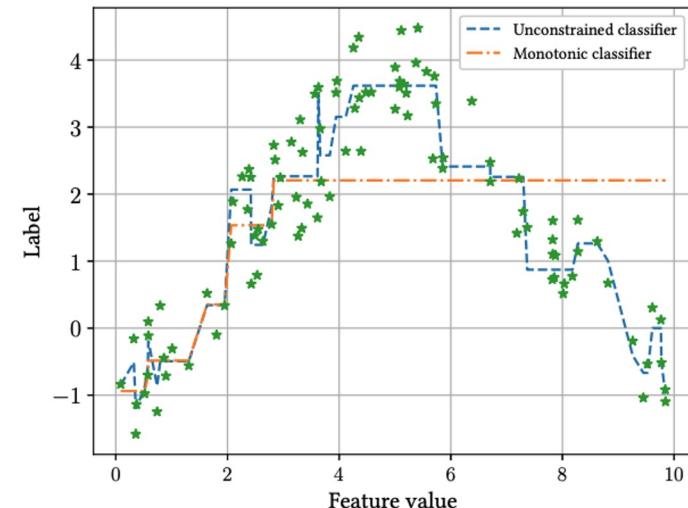
## Adversarially Robust Malware Detection Using Monotonic Classification

Inigo Incer\*  
UC Berkeley  
inigo@eecs.berkeley.edu

Michael Theodorides\*  
UC Berkeley  
theodorides@cs.berkeley.edu

Sadia Afroz  
UC Berkeley  
International Computer Science Institute  
sadia@icsi.berkeley.edu

David Wagner  
UC Berkeley  
daw@cs.berkeley.edu



IWSPA'18, March 21, 2018, Tempe, AZ, USA

Incer et al. *Adversarially Robust Malware Detection using Monotonic Classification*, IWSPA 2018

# Certified Detector

## Certified Robustness of Learning-based Static Malware Detectors

### Formal guarantees

Proofs of non-existence of adversarial examples around input, leveraging edit distance functions.  
Formalized on static end-to-end detectors

### Not reproducible

There are no pre-trained available, nor public source code that can be used to train a model

Zhuoqun Huang  
[zhuoqun@unimelb.edu.au](mailto:zhuoqun@unimelb.edu.au)  
University of Melbourne  
Parkville, VIC, Australia

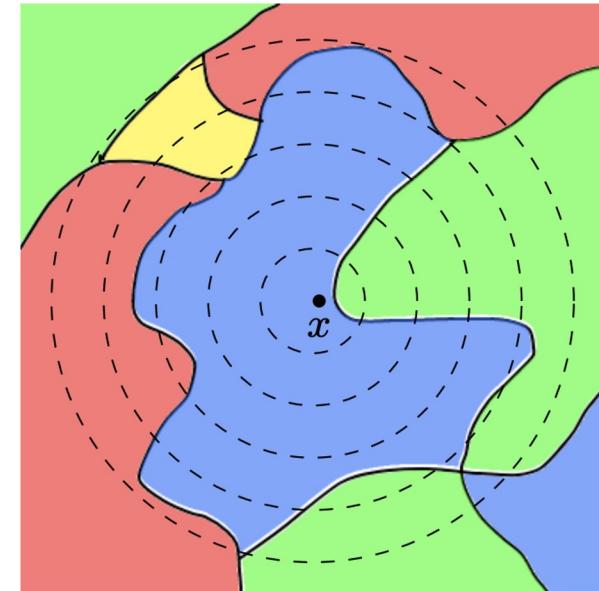
Lujo Bauer  
[lbauer@cmu.edu](mailto:lbauer@cmu.edu)  
Carnegie Mellon University  
Pittsburgh, PA, USA

Neil G. Marchant  
[nmarchant@unimelb.edu.au](mailto:nmarchant@unimelb.edu.au)  
University of Melbourne  
Parkville, VIC, Australia

Olga Ohrimenko  
[oohrimenko@unimelb.edu.au](mailto:oohrimenko@unimelb.edu.au)  
University of Melbourne  
Parkville, VIC, Australia

Keane Lucas  
[keanelucas@cmu.edu](mailto:keanelucas@cmu.edu)  
Carnegie Mellon University  
Pittsburgh, PA, USA

Benjamin I. P. Rubinstein  
[rubinstein@unimelb.edu.au](mailto:rubinstein@unimelb.edu.au)  
University of Melbourne  
Parkville, VIC, Australia



Huang et al. *Certified Robustness of Learning-based Static Malware Detectors*, NeurIPS 2023

# Recent innovation in certification

## Certification for malware

Recent work certifies against padding and content-editing attacks (easier to formalize)

## Specific chunking system

Divide incoming input into chunk according to the format, independently from the size of the file or a fixed number of windows.

## Code available

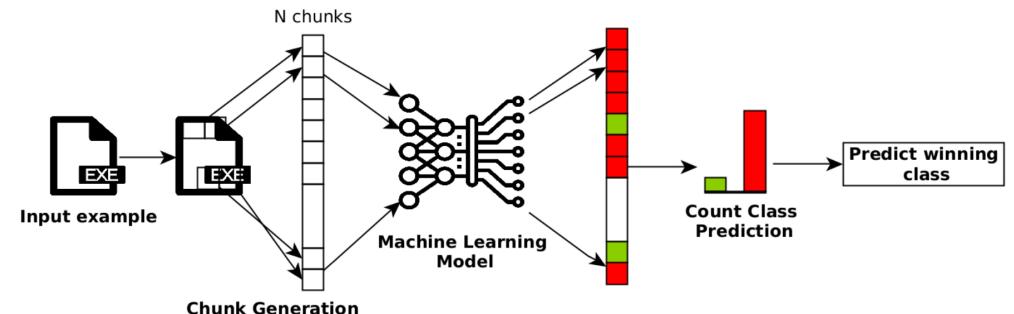
Models can be easily tested, since authors released everything

## Certified Robustness of Static Deep Learning-based Malware Detectors against Patch and Append Attacks

Daniel Gibert  
CeADAR, University College Dublin  
Dublin, Ireland  
daniel.gibert@ucd.ie

Giulio Zizzo  
IBM Research Europe  
Dublin, Ireland  
giulio.zizzo2@ibm.com

Quan Le  
CeADAR, University College Dublin  
Dublin, Ireland  
quan.le@ucd.ie



Gibert et al. *Certified Robustness of Static Deep Learning-based Malware Detectors against Patch and Append Attacks* AISec 2023

## Lab 9:

# Defending against Adversarial EXEmplies

[aka.ms/malware-lab9](http://aka.ms/malware-lab9)

# Limitations and Future Directions

# Issues with Adversarial EXEmpl

## Manipulations are hard to craft

Lack of documentation,  
lack of open-source reference code, lack  
of easily-deployable debugging tools, and  
tons of hours to work

## Evasion is not robustness

In system security, evasion should be  
achieved “no matter what”,  
which is not the same as adversarial  
robustness

# Creating practical manipulations is painful

## Format is vague

Microsoft released vague documentations for the internal of the Windows OS, and research is done by reverse engineering

## PE Format

Article • 06/23/2022 • 127 minutes to read • 15 contributors



This specification describes the structure of executable (image) files and object files under the Windows family of operating systems. These files are referred to as Portable Executable (PE) and Common Object File Format (COFF) files, respectively.

### ⓘ Note

This document is provided to aid in the development of tools and applications for Windows but is not guaranteed to be a complete specification in all respects. Microsoft reserves the right to alter this document without notice.

## Debugging is hard

Manipulations often deal with very specific steps of the loader or runtime execution and it is difficult to get messages from the OS

This app can't run on your PC

To find a version for your PC, check with the software publisher.

Close

# Format specifications are not specific at all

## Many details are omitted

Microsoft released an official format documentation, but it is not complete (as they clearly state)

Example: some header fields are not used by the loader, but they are described as meaningful

## Windows loader changes through time

It has been proven that Windows XP, 7, and 10 have different loaders that parse the PE structure in a different way!

## Closed-source code is not helping

No reference and no code: the only way is either test manipulation by hand, or develop complex tools that infer information about constraints

### ⓘ Note

This document is provided to aid in the development of tools and applications for Windows but is not guaranteed to be a complete specification in all respects. Microsoft reserves the right to alter this document without notice.

## Lost in the Loader: The Many Faces of the Windows PE File Format

Dario Nisi  
EURECOM  
dario.nisi@eurecom.fr

Yanick Fratantonio  
Cisco Talos  
yfratant@cisco.com

Mariano Graziano  
Cisco Talos  
magrazia@cisco.com

Davide Balzarotti  
EURECOM  
davide.balzarotti@eurecom.fr

	Discrepancies				
	W1	W2	W3	W4	W5
XP vs 7	✓	✓			✓
XP vs 10		✓	✓	✓	
7 vs XP					
7 vs 10			✓	✓	
10 vs XP					
10 vs 7		✓			✓

# Complex pipeline for debugging manipulations

## Dealing with kernel components

The building blocks of the operating systems are inside the kernel, there is no easy way to connect them to a debugger.

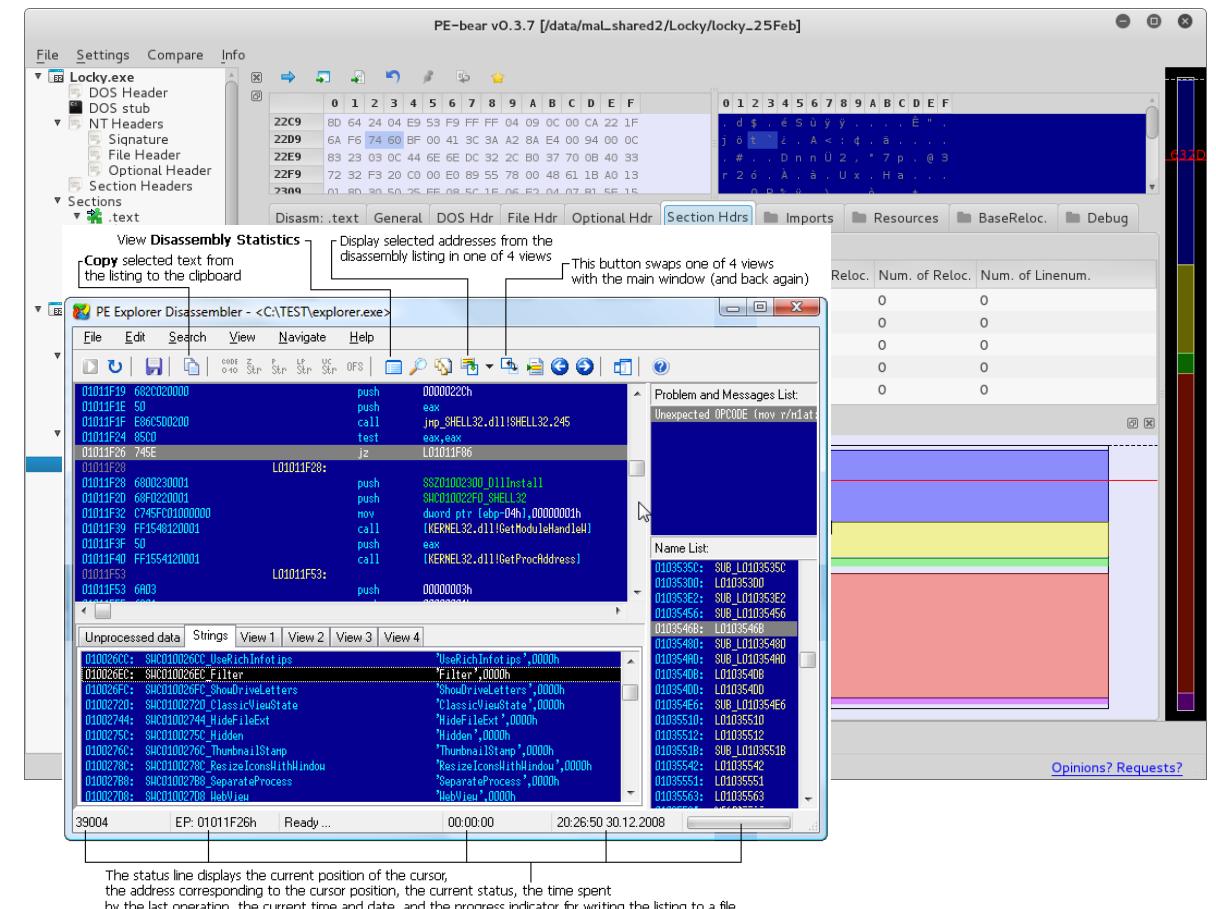
Work-around: manual inspection and tons of wasted hours

## The only output is the error

Perturbed sample is not working? Keep digging without any other informative log, or rely on other PE viewer or checker (PE Bear, PE Explorer, LIEF, pefile...)

## No constraints check

Utilities are good, but they do not tell you IF there is a format specification problem, or they signal vague alerts  
(if you are lucky)



# What about attacks against dynamic classifiers?

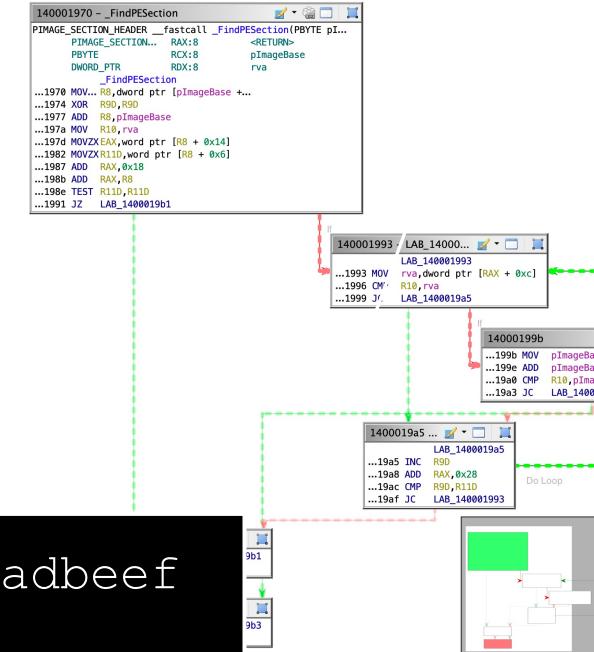
## Not only the structure, the code too!

Code is the only structure that can be manipulated now, so the attacker must act accordingly: code re-writing techniques!

## Different instructions, same functionality

Code is re-written to satisfy properties that the attacker wants, like adding new API calls, invert IF statements, add never-to-be executed code to obfuscate...

```
ADD RAX, 0xdeadbeef  
→  
SUB RAX, -0xdeadbeef
```



# In practice: a nightmare scenario!

## Problems with addresses

If not correctly handled, content injection will shift all known offsets that the compiler created at compile-time

## Problems with executable sections

One could create jumps to other code sections... if they are flagged as executable!

## Problems with relocations

Programs were usually loaded in the same virtual space, but not secure! The OS randomizes the addresses... and this implies that also adversarial content must take this randomization into account!

## Morale: harder than before

It is doable, as there are tons of tools that obfuscate and pack samples, but debugging time increases from a few to many more hours of human work

```
0x7800: MOV EDI, 1  
0x7804: MOV ESI, 2  
0x7808 CALL MY_FUNC  
. . .  
. . .  
0x7880 MY_FUNC
```



```
0x7800: MOV EDI, 1  
0x7804: MOV ESI, 2  
0x7808: XOR EAX, EAX  
0x780a CALL MY_FUNC  
. . .  
. . .  
0x7884 MY_FUNC
```

# Call for action

## Develop attacks against DYNAMIC classifiers

Very little has been done, literally two papers that are not reproducible right now  
Working on this topic will be key point in testing all machine learning malware detectors

Hard? ABSOLUTELY  
Rewarding? ABSOLUTELY

# Evasion is not equal to adversarial robustness

## Two different goals

Evasion implies that malware samples bypass detection, while adversarial robustness quantifies the sensitivity of the detector

## Interested in evasion? Obfuscate & Pack

Tons of literature, open-source code, and material to evade ANY malware detector (with or without machine learning).

Since tools are automatic, it does not change much to perturb or inject few bytes or kilobytes

## Static detection is bypassed by design

Structure of programs can be changed and embedded in other programs, downloaded from the internet after execution, and more



Very well known packer programs that hide malicious content.  
Originally created to prevent reverse engineering of legitimate code.

# Do companies care about Adversarial robustness?

*"We really appreciate this research and would like to collaborate to continue to improve our products and services.*

*At this time this technique does not meet the definition of vulnerability in the product or has demonstrated that it bypassed our products. We will however look into our static ML models to see how we can incorporate this technique to further improve."*

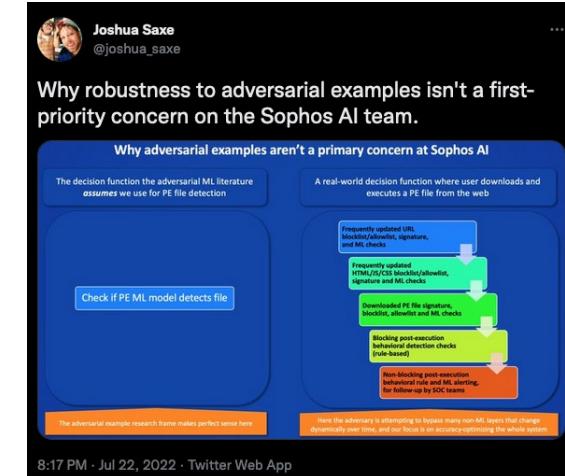
One-of-those-company

## Adversarial EXEmplEs not treated as vulnerability

Companies are interested in evading the overall pipeline, not just portions of the products

Naïve solution: test attacks against deployed commercial products

(which are unavailable, as said before)



## Missing the bigger picture

Companies have the feelings that academic settings are unrealistic, as they target "only" the ML component

# Call for action

## Develop new testing techniques that consider all components

We are starting to create end-to-end pipelines, but we are still missing a complete framework that systematically tell a developer HOW to test these models

# Part 2 Survey

[aka.ms/malware-survey2](http://aka.ms/malware-survey2)