SmartLab

Pattern Recognition and Applications Lab

Adversarial EXEmples: Evading Windows Malware Detectors

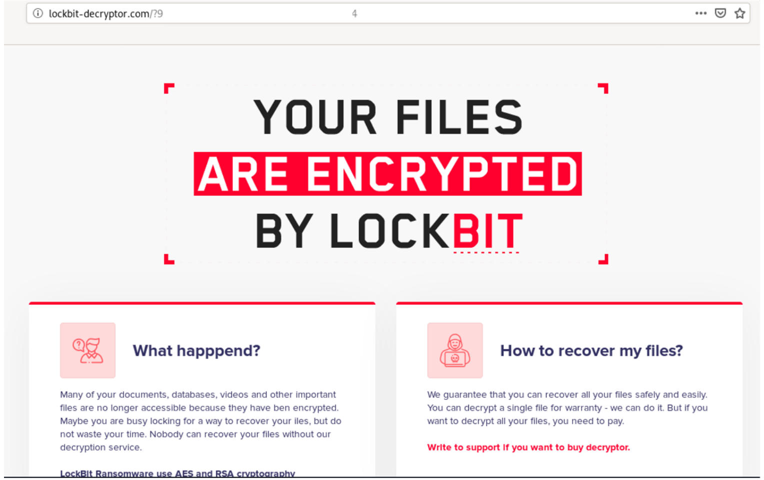
Luca Demetrio
luca.demetrio@unige.it
@zangobot

Part 1 – Malware and how to detect it

Malware: dangerous programs

Malicious by design

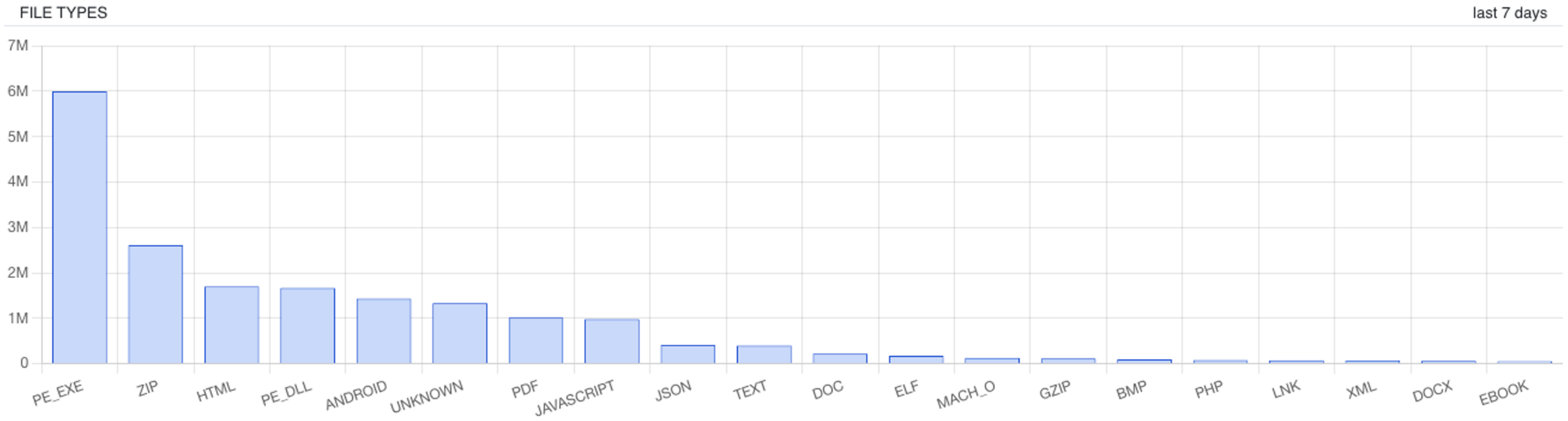
Intrusive programs whose intent is causing damage, steal information, ransom, take control of devices



"Business" of malware

Currently, there are plenty of famous active groups that act as companies that produce and sell malware as a service

Concerning numbers



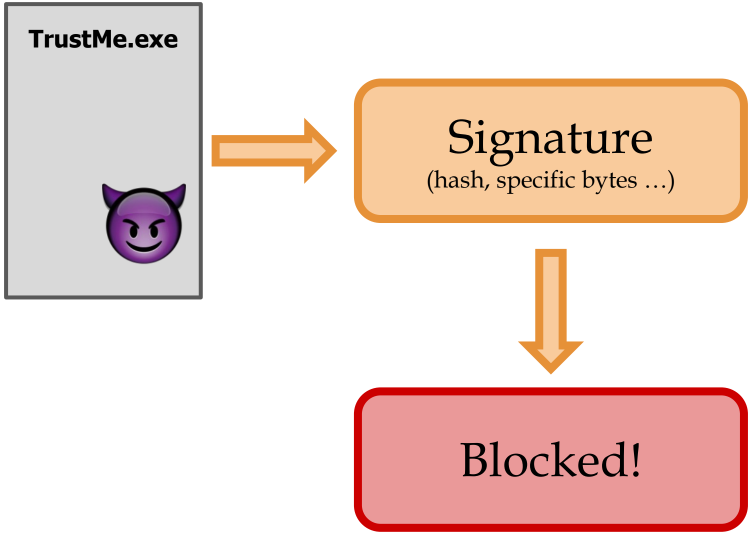
Every week, **6M** of Windows executable are scanned with cloud systems to detect malware, with more than **1M** detected samples in the same timespan

[VirusTotal, https://www.virustotal.com/gui/stats]

Security *without* Machine Learning

Blocklist approach

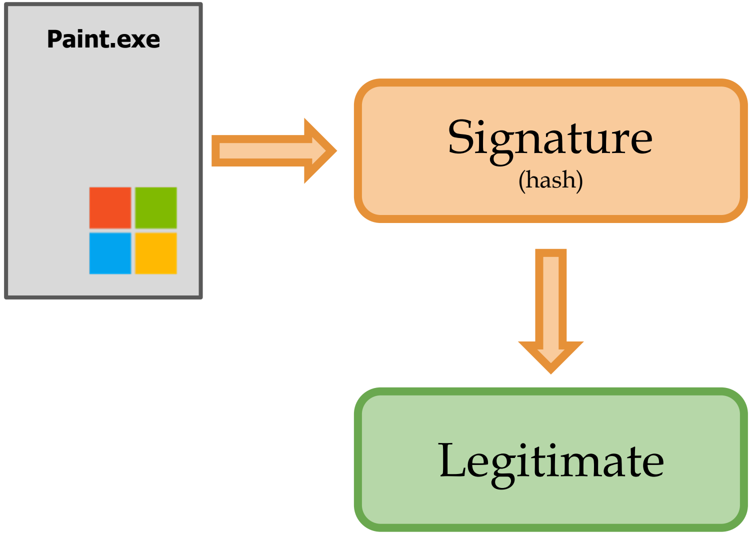
1. Extract a "**signature**" from each incoming request, like hashes, the presence of specific bytes or word (like "Viagra" for spam)
2. If these signatures are contained inside well-known **blocklists,** the input is recognized as **malicious**



Security *without* Machine Learning

Allowlist approach

1. Extract a "**signature**" from each incoming request, like hashes.
2. If these signatures are contained inside well-known **allowlists,** the input is recognized as **harmless**, otherwise it is blocked



Why not black/white list?
https://www.ncsc.gov.uk/blog-post/terminology-its-not-black-and-white

YARA Rules

Known patterns as text

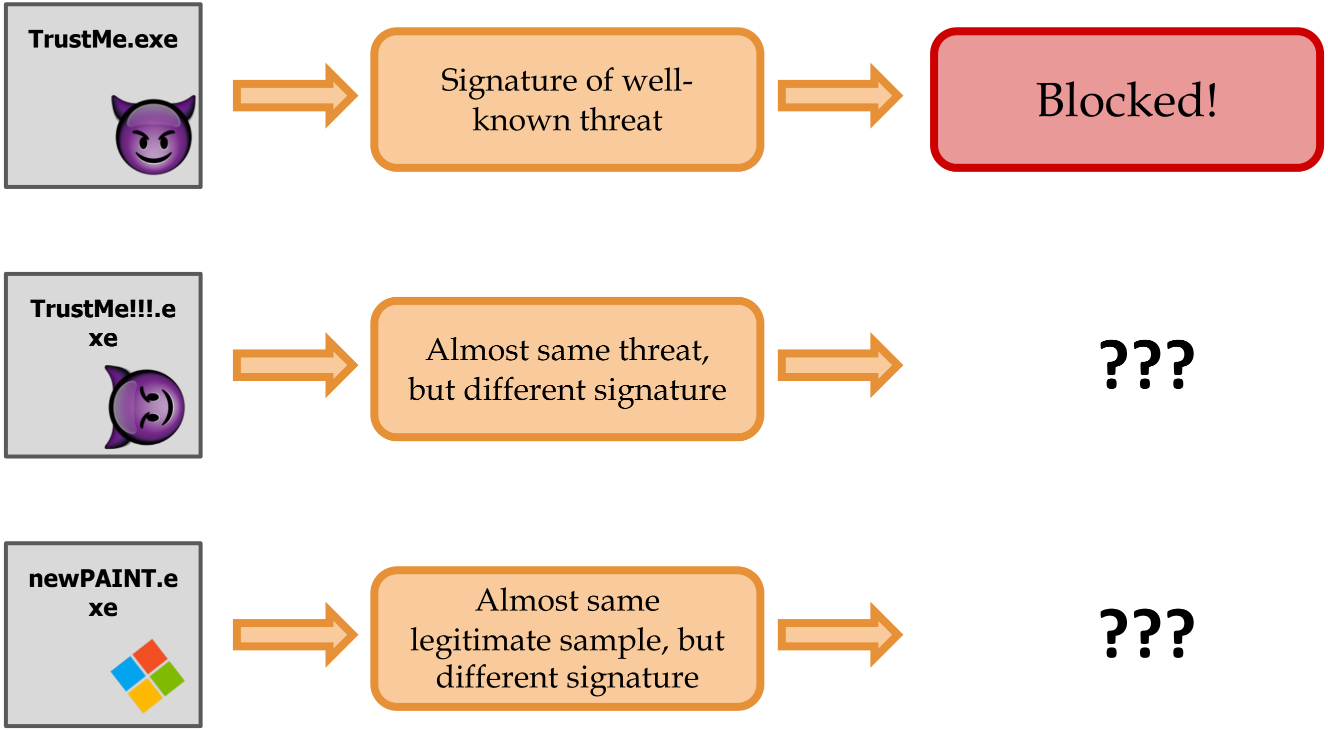
A rule is a sequence of bytes, and files match depending on the specified condition

Plenty of open-source rules

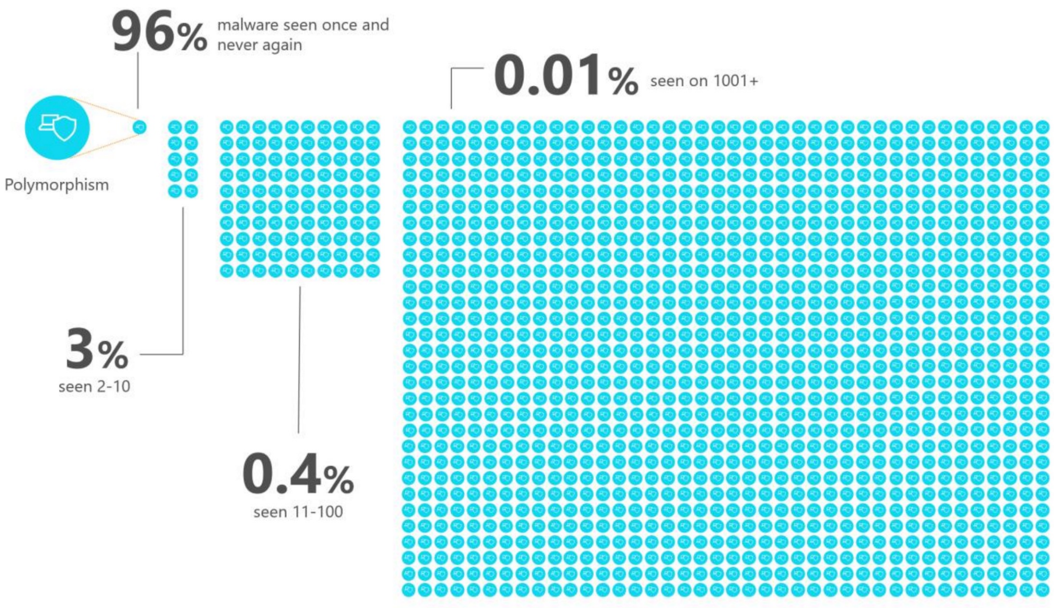
Try to google “YARA rules” :D

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

Limitation: what happens with minimal changes?



Required amount of static signatures



Too many variants!

Block/Allowlist can work ONLY if threats are known, but the majority of them (96%) appear in the wild only **ONE** time

Evolution of Malware Prevention
[https://info.microsoft.com/rs/157-GQE-382/images/Windows%2010%20Security%20Whitepaper.pdf]

Suboptimal performances

Quality of rules matters

Open-source rules are not the best around, and it is possible that companies possess better ones for detection

(Thanks to Andrea Ponte, Ph.D. @ SMARTLAB for the initial analysis!)

```
Compiled rules:2563
Tested malwares:5005
Detected malwares: 765
Total time taken: 69.11032915115356
Average time taken: 0.013808257572658056
Standard deviation: 0.038165608714924255
```

Machine Learning for Malware Detection

Machine Learning applied as Security Scanner

Spreading into commercial products
Companies claim to use machine learning technologies inside their detectors to spot Windows malware by learning patterns from data

Filter out known threats, generalize to variants
Deep networks learn "signatures", and they can spot variants of the same malware

SOPHOS
Security made simple.

Intercept X Deep Learning

CrowdStrike Introduces Enhanced Endpoint Machine Learning Capabilities and Advanced Endpoint Protection Modules

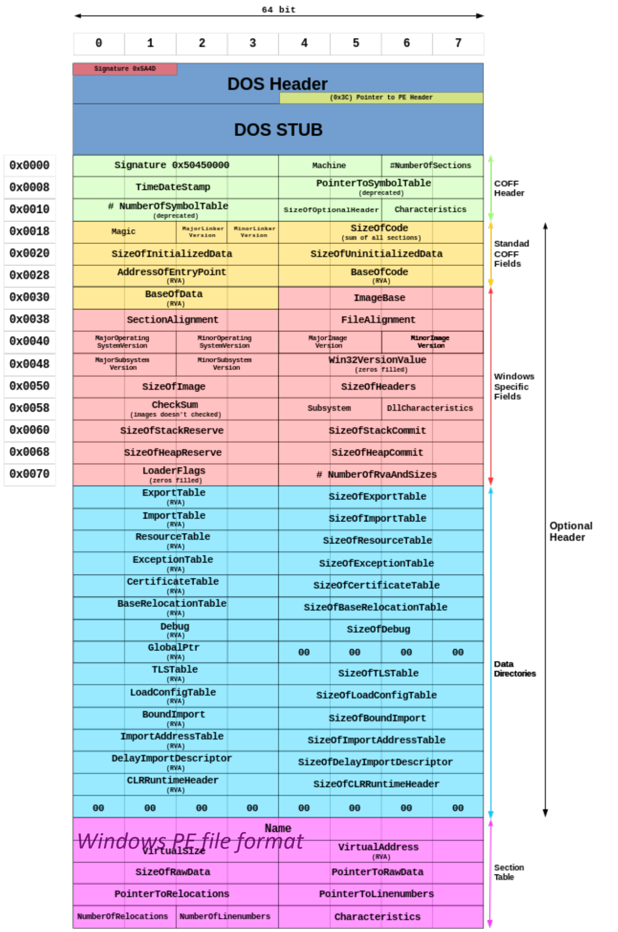
— Company continues to accelerate pace of replacement of legacy AV solutions in both enterprise and SMB markets —

Static malware detection with ML

Programs stored as file
Each program is a regular file that can be analysed without executing it

Features to extract
Which API they import? How many resources they contain? Are there some initialized values? Do they require special permissions?

End-to-end learning
Or, use each byte of the program/resources/files as token, let the network learn by itself a suitable representation

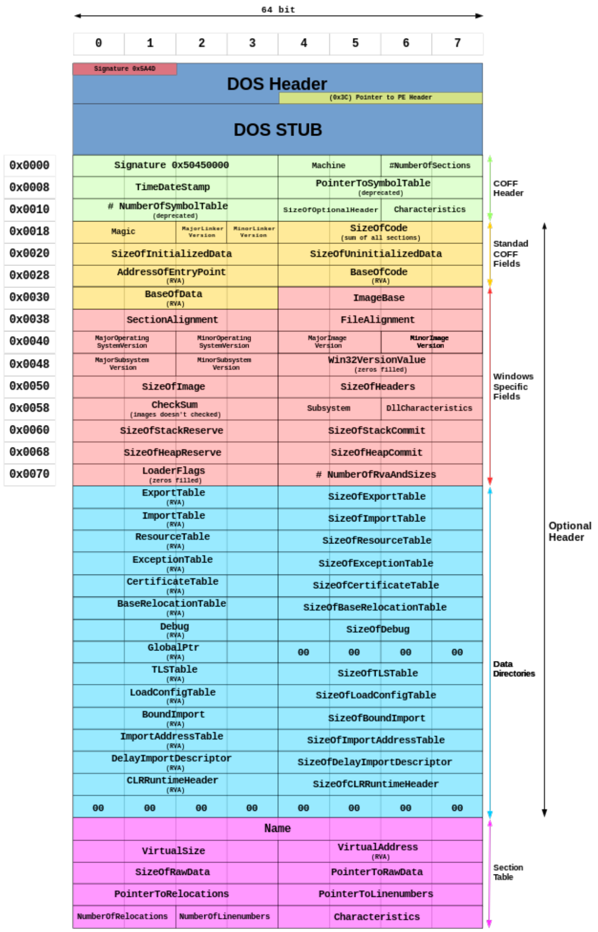


Windows PE file format

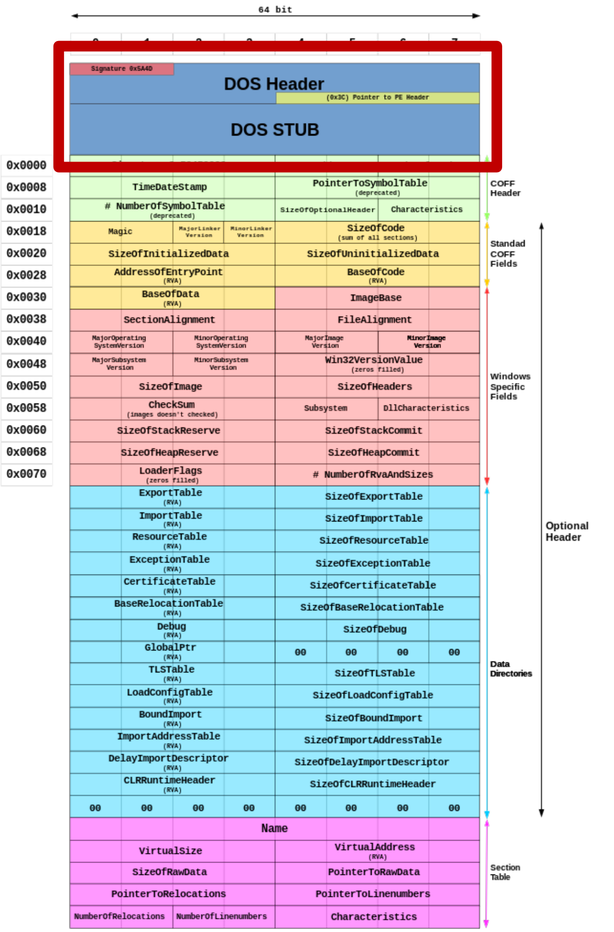
Windows PE File Format

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

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



Windows PE File Format



DOS Header + Stub

Metadata for DOS program

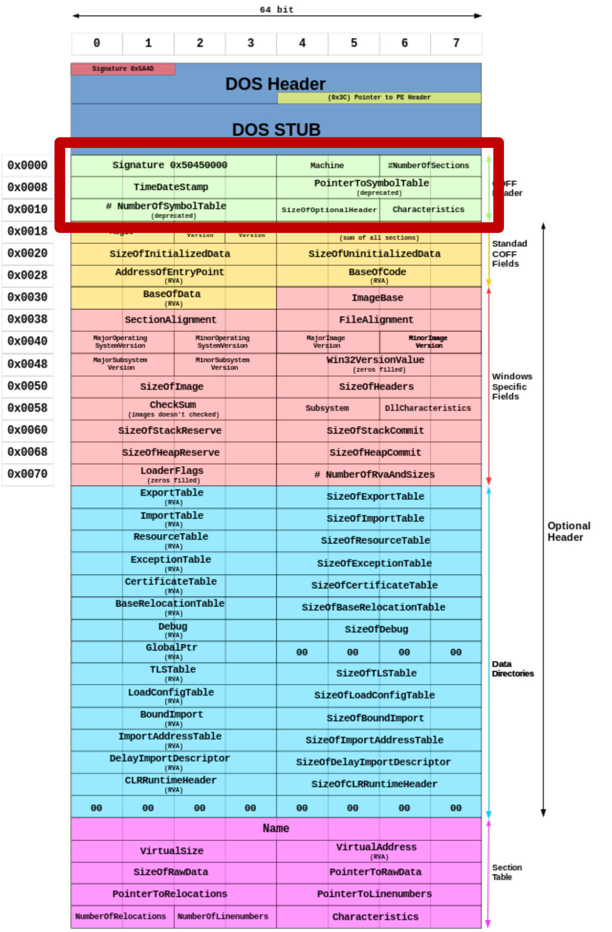
Executing a modern program in DOS will trigger the “This program cannot be run in DOS mode” output

Windows PE File Format

PE Header

Real metadata of the program

Dscribes general information of the file

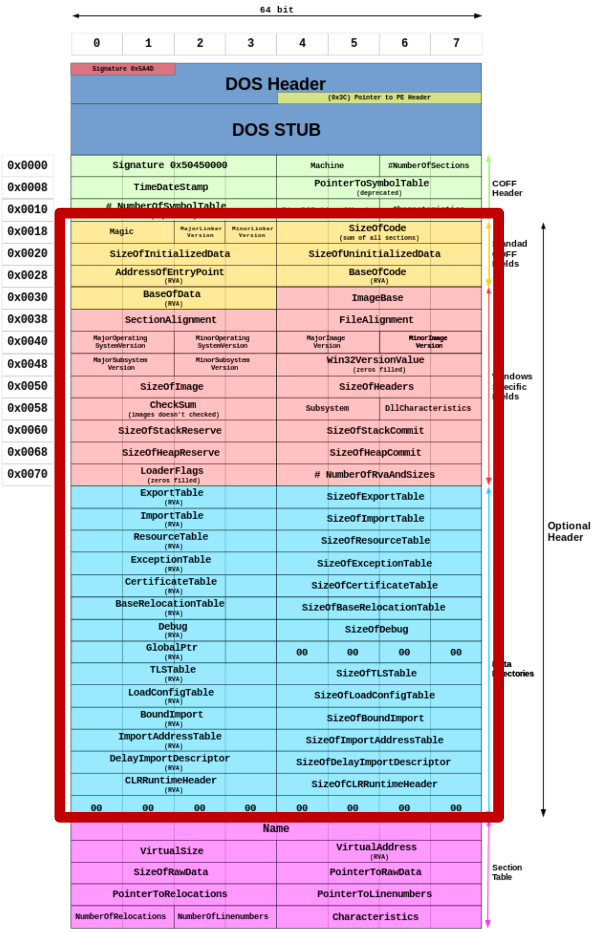


Windows PE File Format

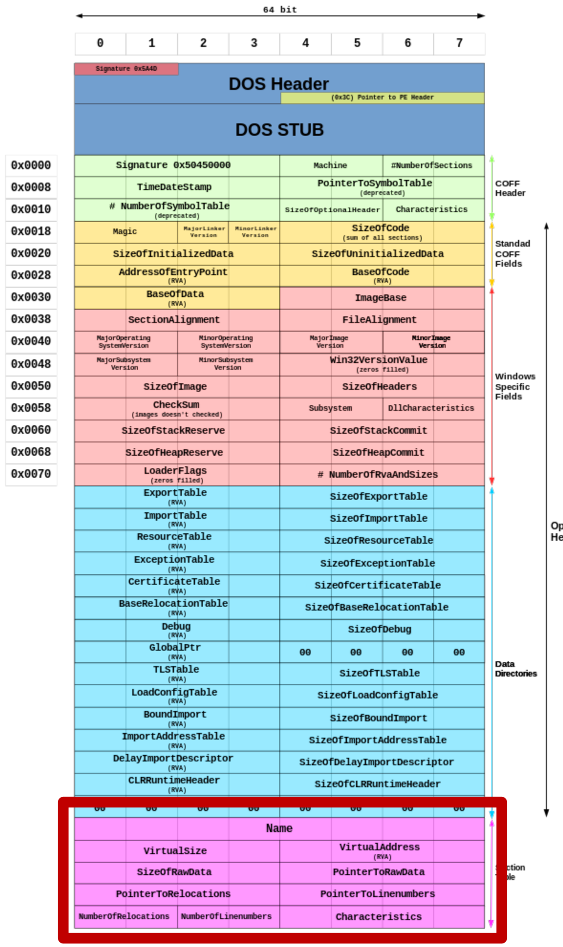
Optional Header

Spoiler: not optional at all :)

Instructs the loader where to find each object inside the file



Windows PE File Format



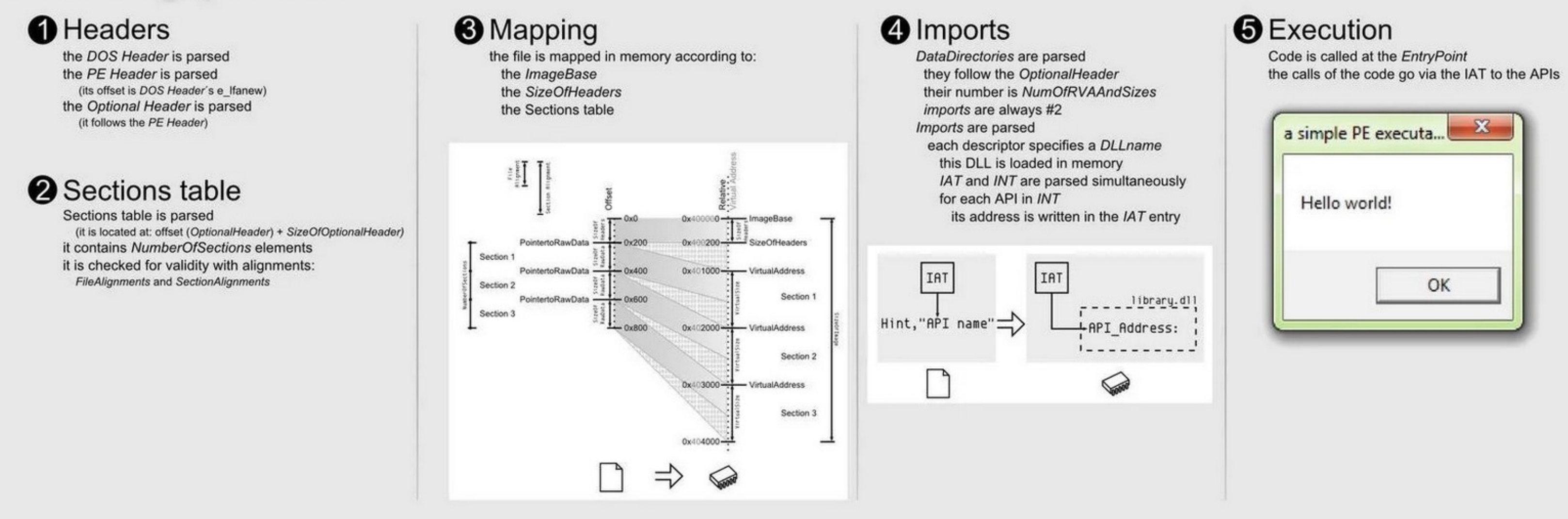
Section Table and Sections

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

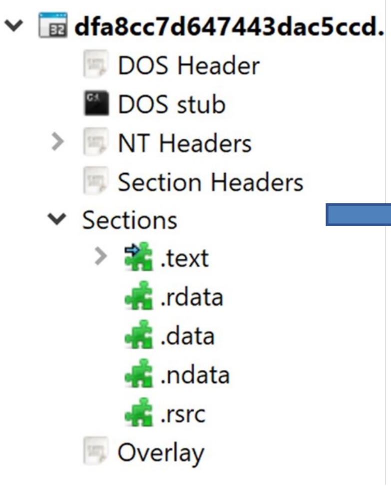
How programs are loaded



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

(Thanks Ange Albertini)

Static malware detection with ML



Static features (import, exports, sections, bytes, …)

Supervised ML

p(malicious) = 0.951

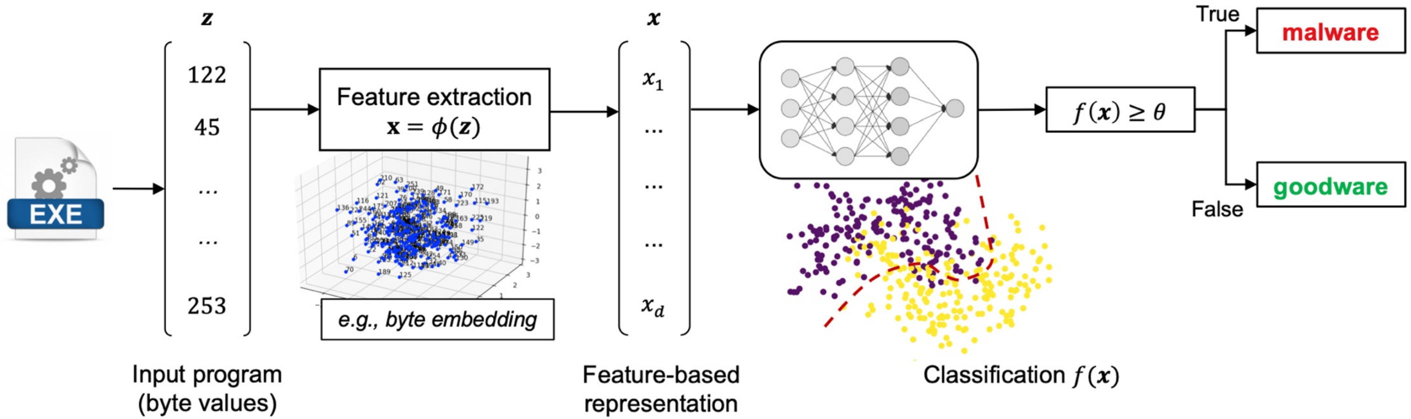
Human or "greedy" labels:

Malicious

Benign

[Thanks to Dmitrijs Trizna for these slides!]

Examples of Static Windows Malware Detectors



Raw bytes as features

MalConv => **1 MB** in input, embedding

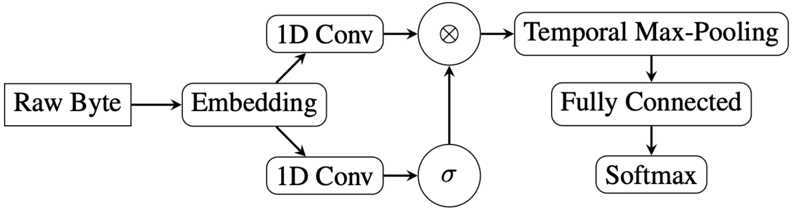
Static features from data

GBDT => decision tree on **2.381** features
(API, sections, byte entropy, exports, strings, etc.)

[Anderson et al. *EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models*, arXiv preprint 2018]
[Raff et al., *Malware Detection by Eating a Whole EXE*, AAAI Workshop 2018]

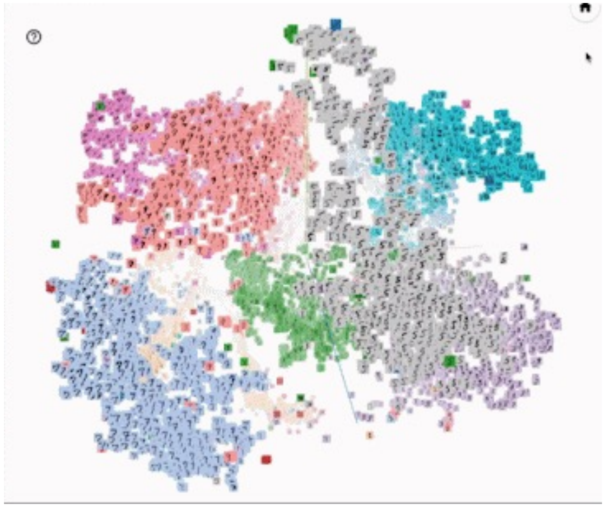
MalConv: end-to-end classification

Total params to train: **1.042.953**



Embedding layer
Each byte is converted in a 8-dimensional vector learned at training time.

Convolutions similar to images
Analyse 1D patterns of bytes to retrieve local information, later aggregated by the fully-connected layer



(Thanks to Dmitrijs Trizna for the animation!)

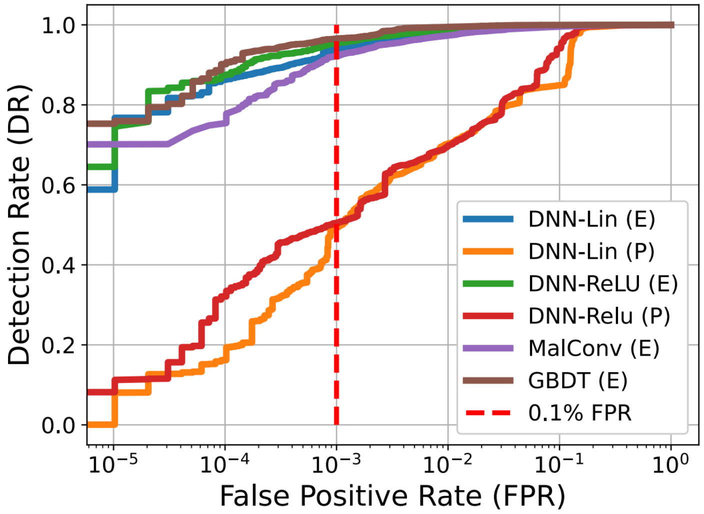
GBDT EMBER: classic feature extraction

Total params to train: **2381**

Hand-crafted features
Sections, histogram of bytes, distribution of strings, API calls, etc

Harder to develop, faster to test
Instilling domain knowledge is hard, but the gradient-boosting decision tree model is fast to train (faster than a neural network)

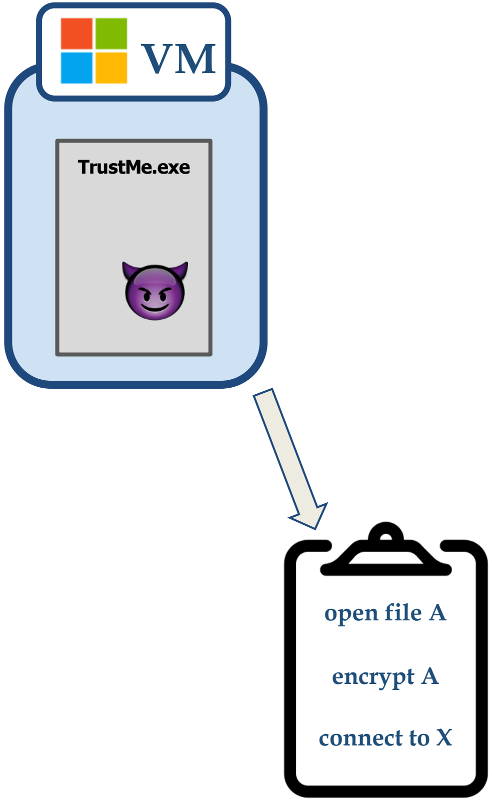
Performance comparison



Hand-crafted features seem better

Computed on the EMBER dataset, GBDT exhibits superior performance to other models

Dynamic malware detection with ML



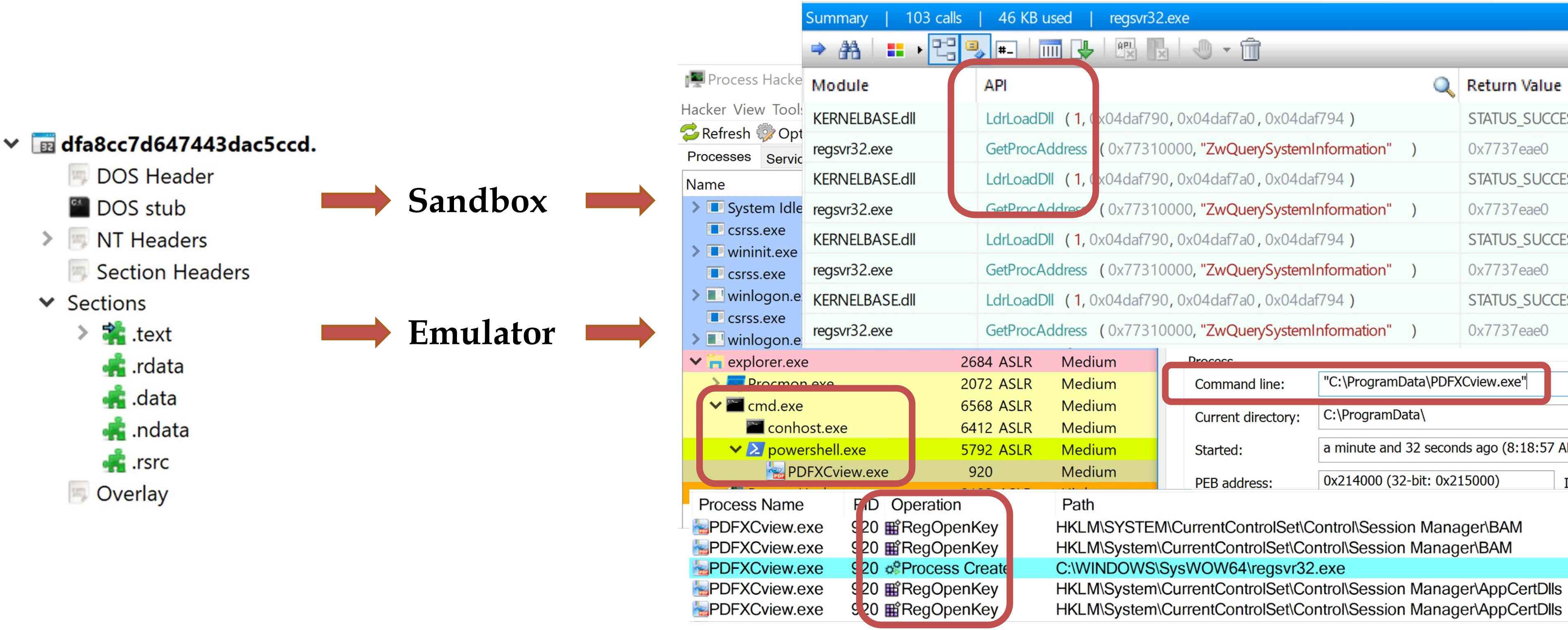
Chain of events
Run program inside protected isolated environment, take note of every observable action of the program

Human-readable reports
The analysis outputs a textual report that specifies the timeline of all the triggered events

Circumventing obfuscation
Even if samples are packed or obfuscated, at some point the functionality will be manifested through interactions with the underlying OS

[Thanks to Dmitrijs Trizna for these slides!]

Dynamic malware detection with ML

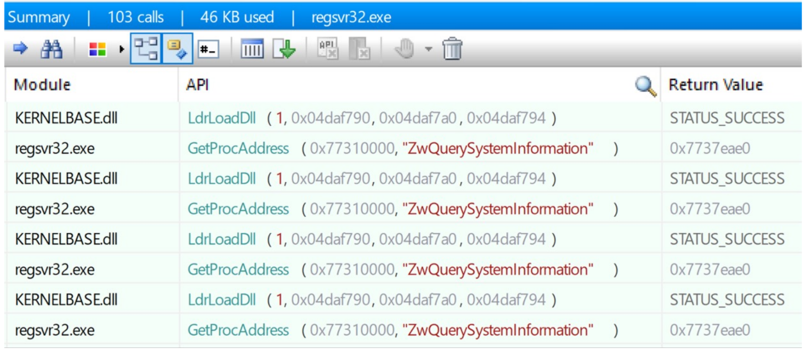


Dynamic malware detection with ML

Traces of execution
Structure alone can’t reveal too much: exploit behavior of analysed programs
Running programs becomes necessary

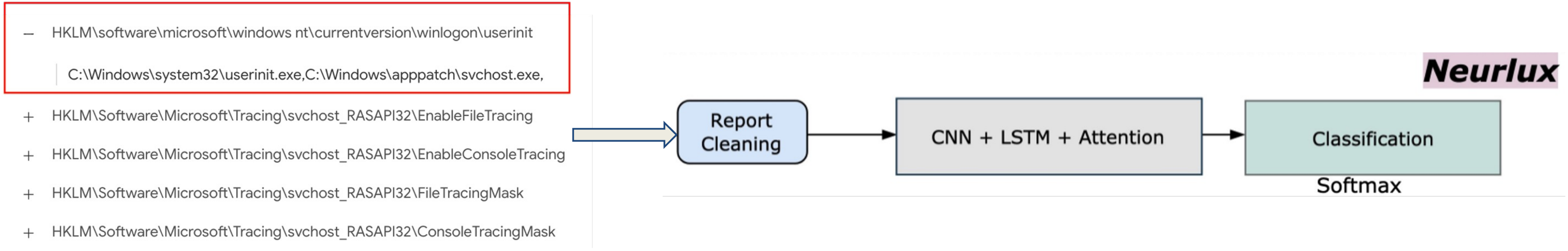
Features to extract
Which API they call? Which IP addresses they contact? Which service they interact with?

End-to-end learning
Or, consider the sequence of actions as a list of token that can be used in NLP systems


Summary | 103 calls | 46 KB used | regsvr32.exe

Module	API	Return Value
KERNELBASE.dll	LdrLoadDll (1, 0x04daf790, 0x04daf7a0, 0x04daf794)	STATUS_SUCCESS
regsvr32.exe	GetProcAddress (0x77310000, "ZwQuerySystemInformation")	0x7737eae0
KERNELBASE.dll	LdrLoadDll (1, 0x04daf790, 0x04daf7a0, 0x04daf794)	STATUS_SUCCESS
regsvr32.exe	GetProcAddress (0x77310000, "ZwQuerySystemInformation")	0x7737eae0
KERNELBASE.dll	LdrLoadDll (1, 0x04daf790, 0x04daf7a0, 0x04daf794)	STATUS_SUCCESS
regsvr32.exe	GetProcAddress (0x77310000, "ZwQuerySystemInformation")	0x7737eae0
KERNELBASE.dll	LdrLoadDll (1, 0x04daf790, 0x04daf7a0, 0x04daf794)	STATUS_SUCCESS
regsvr32.exe	GetProcAddress (0x77310000, "ZwQuerySystemInformation")	0x7737eae0

Example of Dynamic Windows Malware Detector

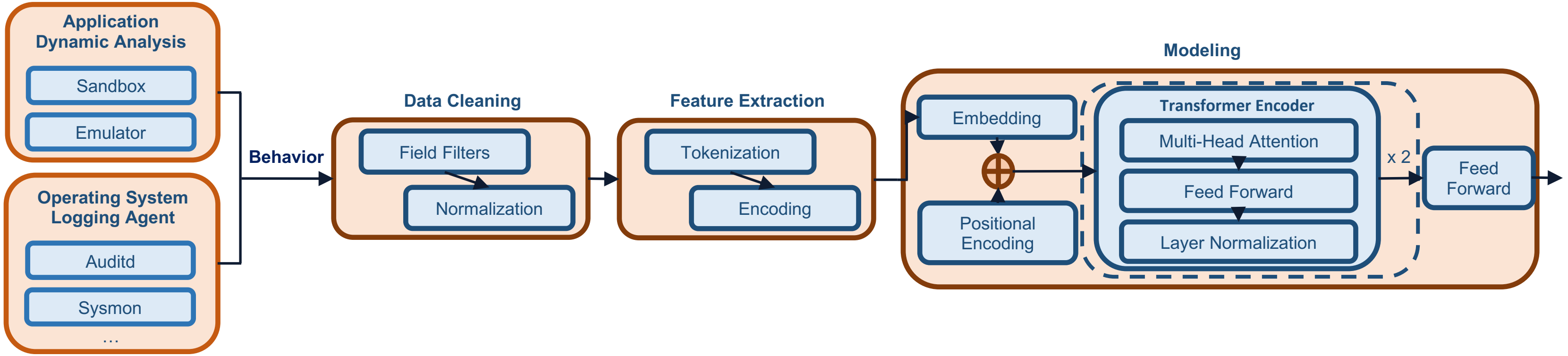


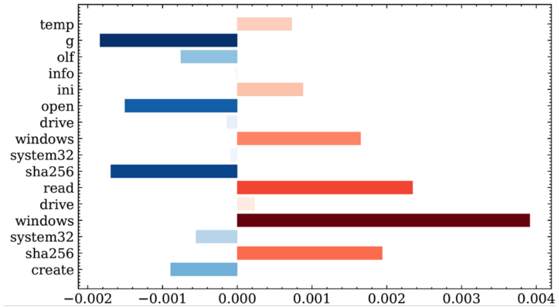
Reports as input file

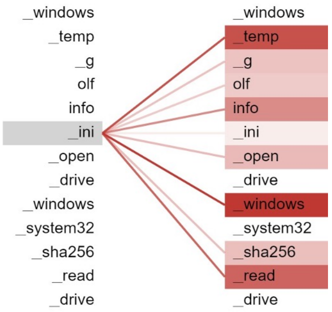
Neurlux takes in input entire textual report, divided in tokens fed to deep neural networks / transformers

[Jindal et al. *Neurlux: dynamic malware analysis without feature engineering*, ACSAC 2019]

NEBULA: Self-attention for Dynamic Malware Analysis







Learning the language of malware

Joint work to understand how malware behaves from reports, leveraging cutting-edge transformers that outperforms other models in NLP (thank you Dmitrijs Trizna, national AI Ph.D. student & SSR @ Microsoft)

[Trizna, Demetiro, Biggio et al. *NEBULA: Self-attention for Dynamic Malware Analysis*, preprint 2023]

Static or dynamic?

Static > dynamic
Recent work show how much dynamic analysis is prone to error and noisy, opposed to the static one

… but dynamic better on unknowns
Static analysis perform worse on unseen families, while dynamic seems to be a bit better

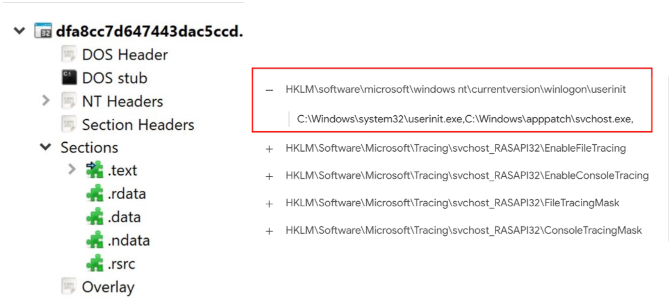
Static + dynamic?
The combination is not improving much the performances of static classifiers alone

Decoding the Secrets of Machine Learning in Windows Malware Classification: A Deep Dive into Datasets, Features, and Model Performance

Savino Dambra* — Norton Research Group
Yufei Han* — INRIA
Simone Aonzo* — EURECOM
Platon Kotzias* — Norton Research Group
Antonino Vitale — EURECOM
Juan Caballero — IMDEA Software Institute
Davide Balzarotti — EURECOM
Leyla Bilge — Norton Research Group

[Dambra et al. *Decoding the Secrets of Machine Learning in Windows Malware Classification: A Deep Dive into Datasets, Features, and Model Performance*, CCS 2023]

What about the best of ALL world?



Missing composite solution
Currently no proposal from the literature for complete AI systems that detect malware using rules, static, and dynamic analysis (also to mimic industry settings)

How to connect them?
While there are study that show that dynamic analysis improve accuracy by only few percentage points, there are still no clues on how to compose these layers

On-going work
We are investigating the performance of these AI systems to also improve the quality of testing (Thanks to Andrea Ponte again!)

[Jindal et al. *Neurlux: dynamic malware analysis without feature engineering*, ACSAC 2019]

Take-home messages of Part 1

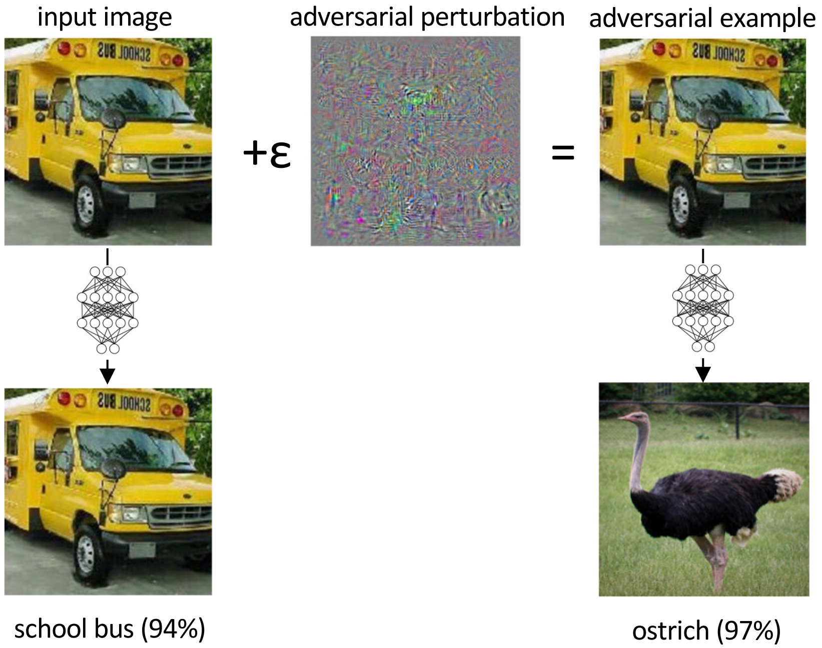
Machine learning helps in the fight
Learning from data generalizes better than collecting rules and hashes

Domain knowledge vs end-to-end
Data can be used as-is or with feature extraction, and the second performs better

Static vs Dynamic
Different ways to recognize malicious patterns, either from the structure of the file or from its behavior. Recent study shows that static is better.

Part 2 - Adversarial EXEmples

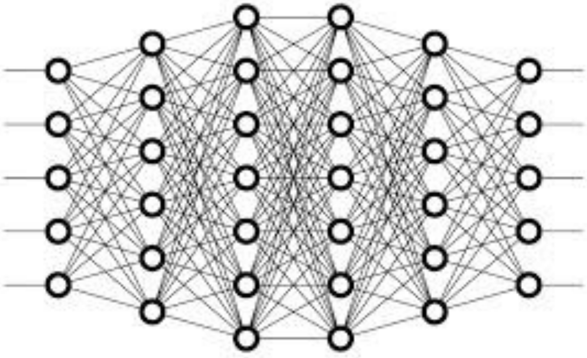
Adversarial Examples (Gradient-based Evasion Attacks)

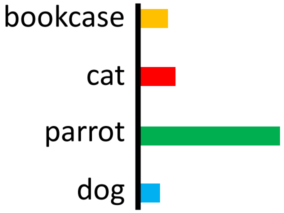


(Thanks to Battista Biggio, Maura Pintor for these slides!)

Szegedy et al., Intriguing properties of neural networks, **ICLR 2014**
Biggio, Roli, et al., Evasion attacks against machine learning at test time, **ECML-PKDD 2013**

Adversarial Attacks



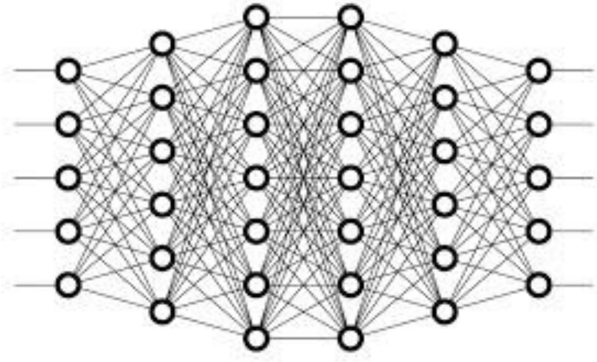


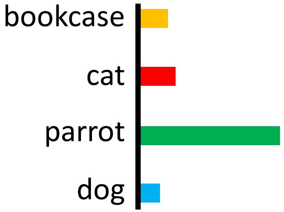
Adversarial attacks exploit the same underlying mechanism of learning, but aim to maximize the probability of error on the input data: $\max_D L(D; \boldsymbol{w})$

This problem can also be solved with gradient-based optimizers
(Biggio, Roli et al., ICML 2012; Biggio, Roli et al., ECML 2013; Szegedy et al., ICLR 2014)

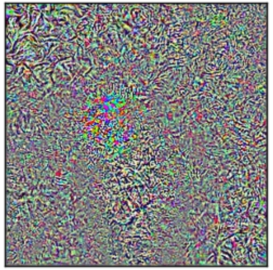
(Thanks to Battista Biggio, Maura Pintor for these slides!)

How Do These Attacks Work?





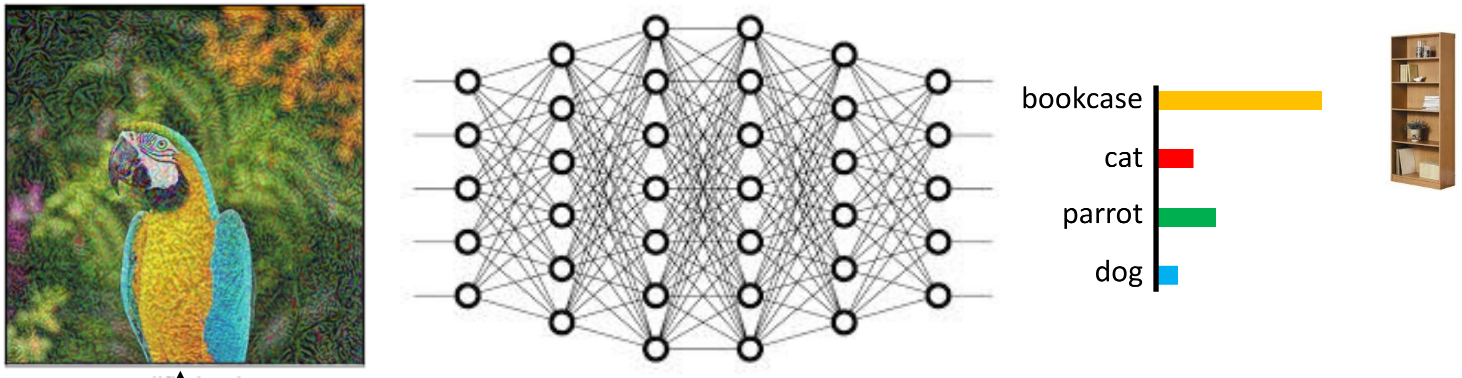
$$\max_{D} L(D; \boldsymbol{w})$$

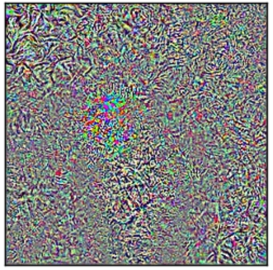


The gradient of the objective allows us to compute an *adversarial perturbation…*

(Thanks to Battista Biggio, Maura Pintor for these slides!)

How Do These Attacks Work?

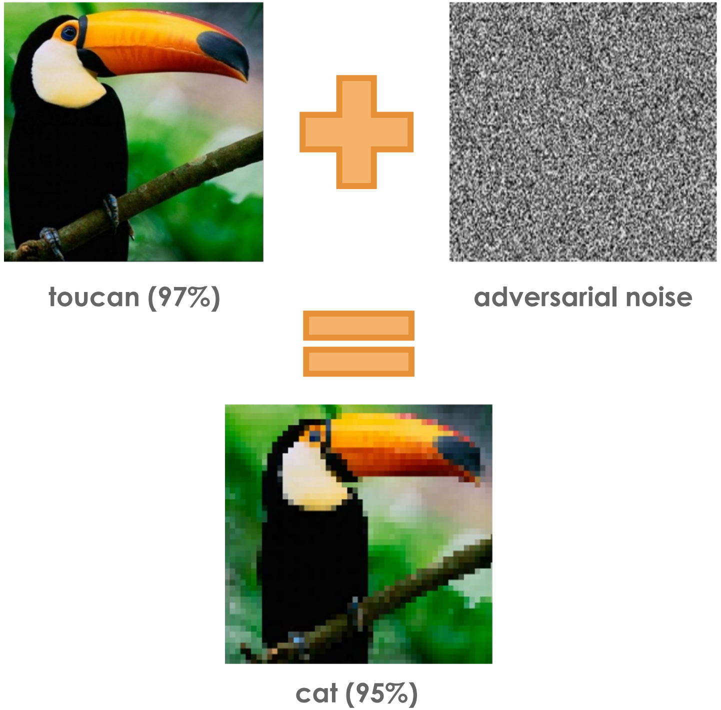


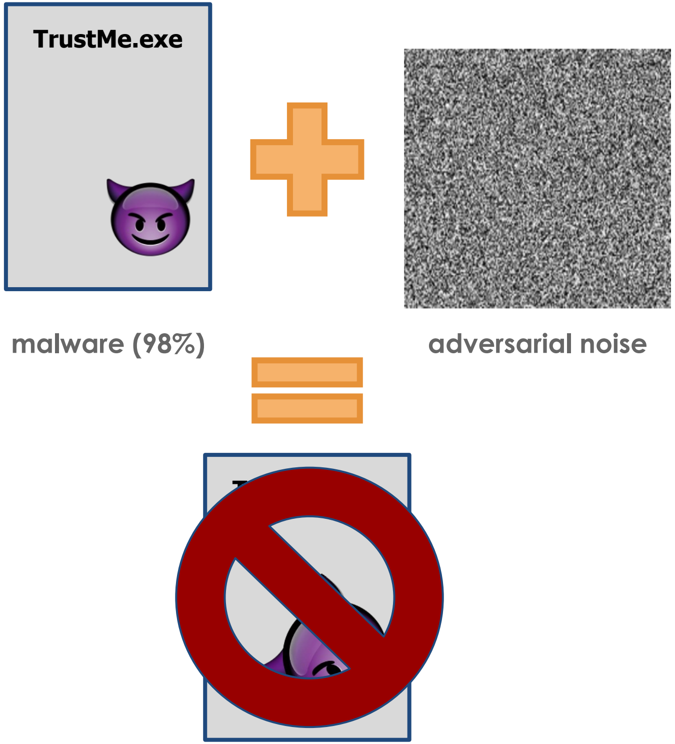


... which is then added to the input image to cause misclassification

(Thanks to Battista Biggio, Maura Pintor for these slides!)

Same for malware?





Not runnable anymore!

Byte-sequences are not numbers

Programs and images are encoded in bytes

RGB is "continuous", code instructions are not!

Distance between programs is **undefined**

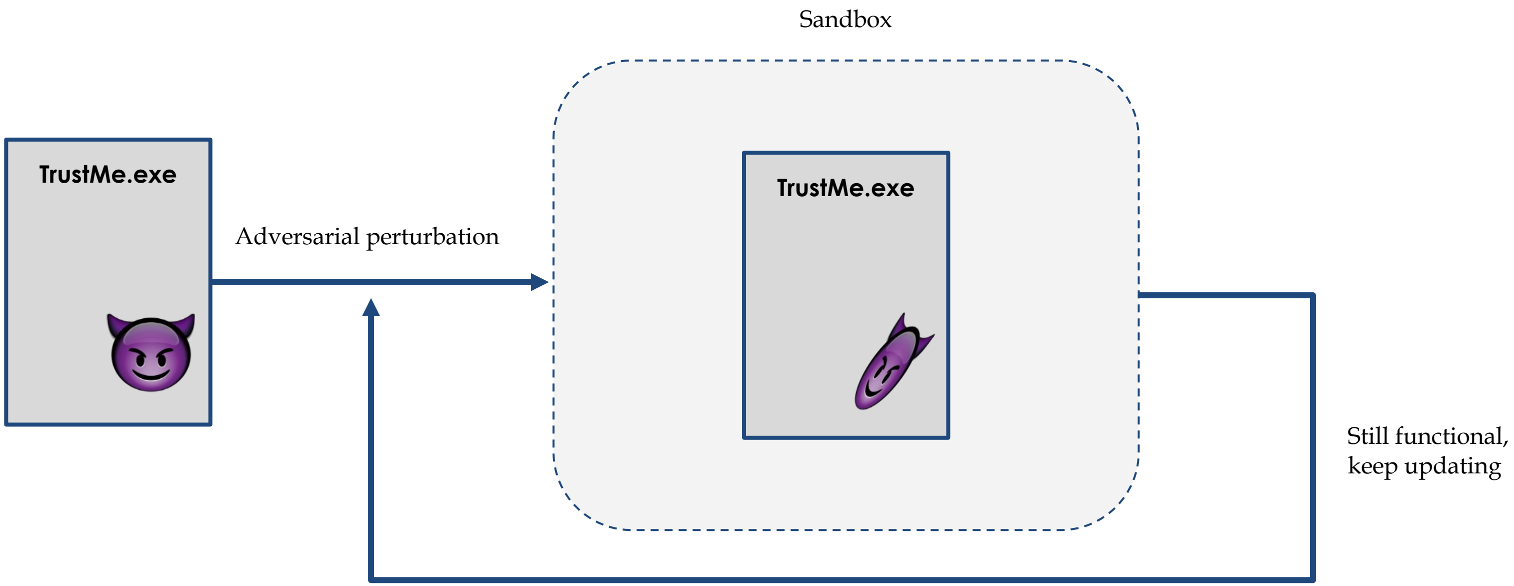
Example: ASCII table

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		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

Preserve the original functionality



Song et al., *MAB-Malware: A Reinforcement Learning Framework for Attacking Static Malware Classifiers*, arXiv preprint 2021
Castro et al., *AIMED: Evolving Malware with Genetic Programming to Evade Detection*, TrustCom/BigDataSE 2019

How to bridge these gaps?

1. Formulate the minimization problem differently
2. Study the format that represent programs
3. Understand how to exploit the format
4. Chose how to inject or perturb the content

Formulation of the problem

Adversarial attacks for images

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

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

Adversarial attacks for security detectors

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

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 EXEmples: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

Take-home message: implementing an attack

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

Define the Optimizer
Depending on the differentiability of the compontens, 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

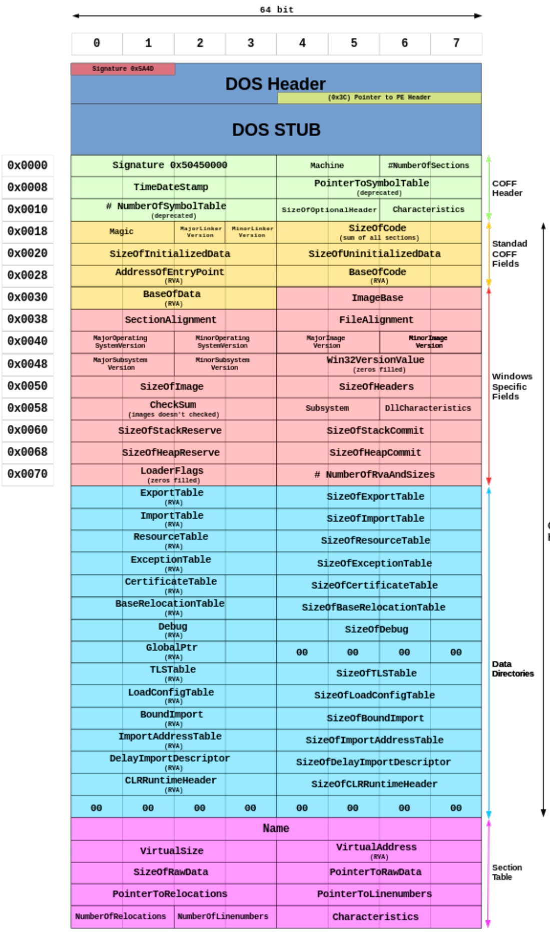
Demetrio et al., *Adversarial EXEmples: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

Manipulations of Windows PE file format

Windows PE File Format

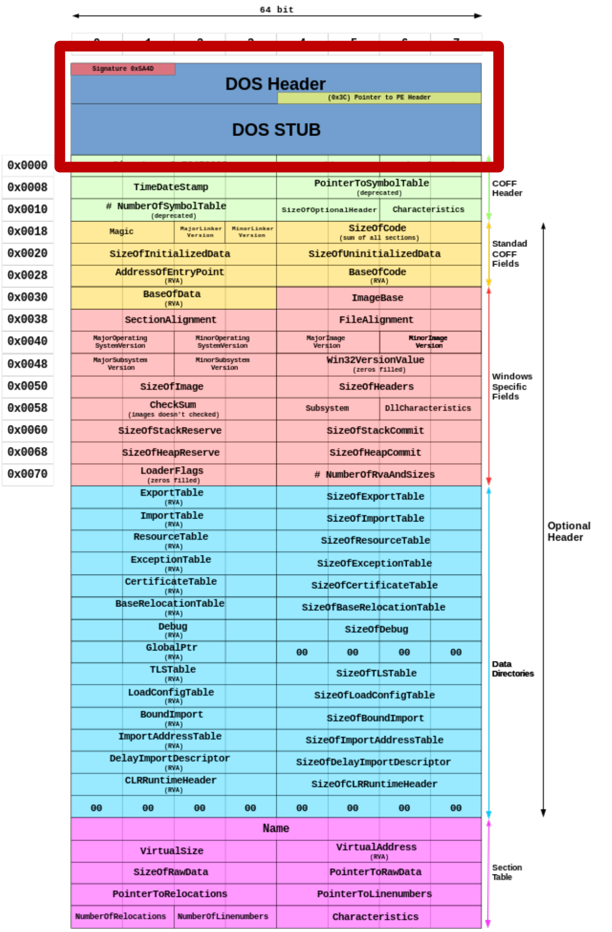
Format adapted for "modern" programs (from Windows NT 3.1 on)

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



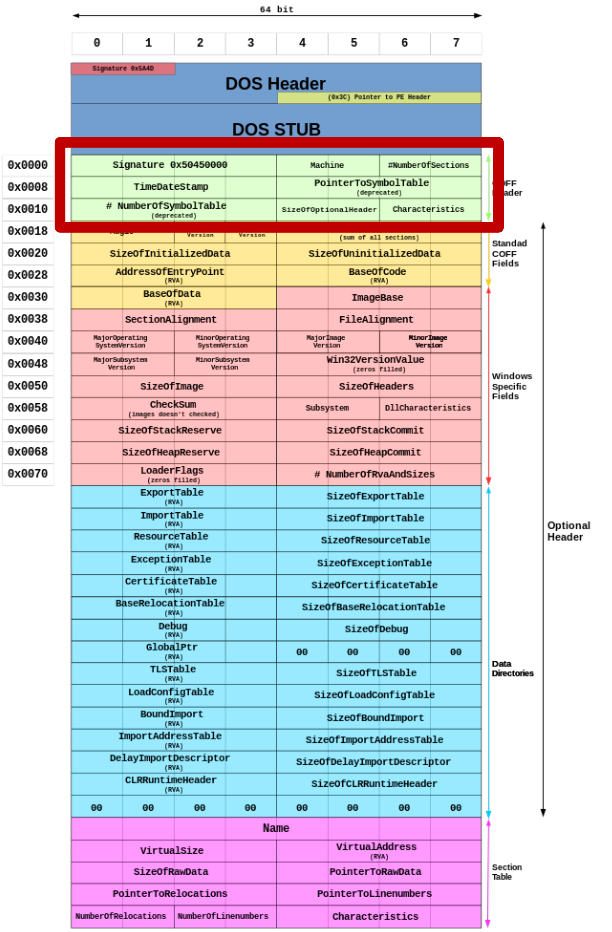
Windows PE File Format

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



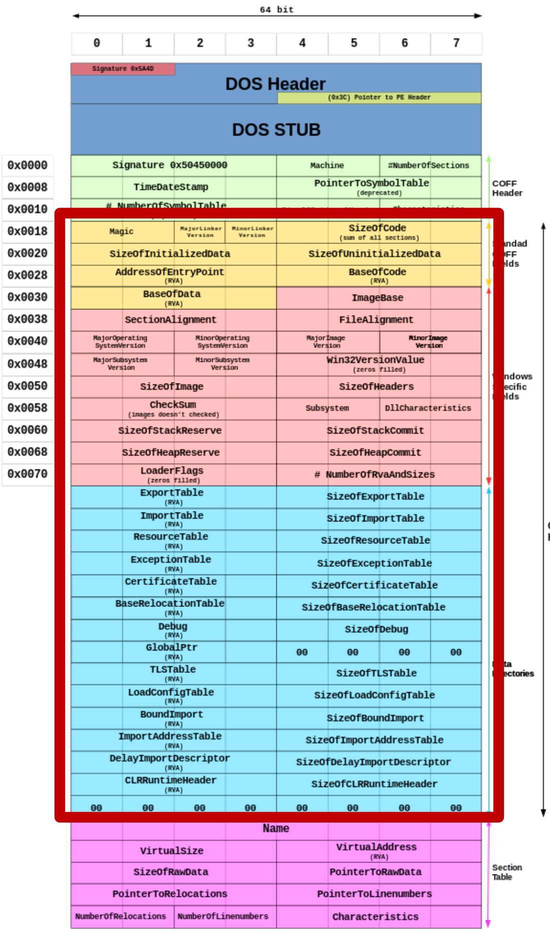
Windows PE File Format

PE Header
Real metadata of the program
Dscribes general information of the file



Windows PE File Format

Optional Header
Spoiler: not optional at all :)
Instructs the loader where to find each object inside the file



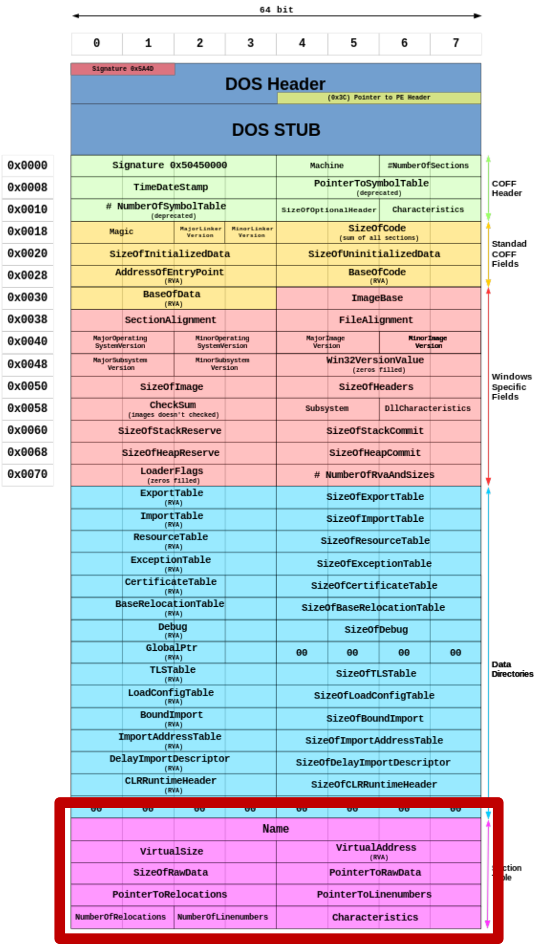
Windows PE File Format

Section Table and Sections

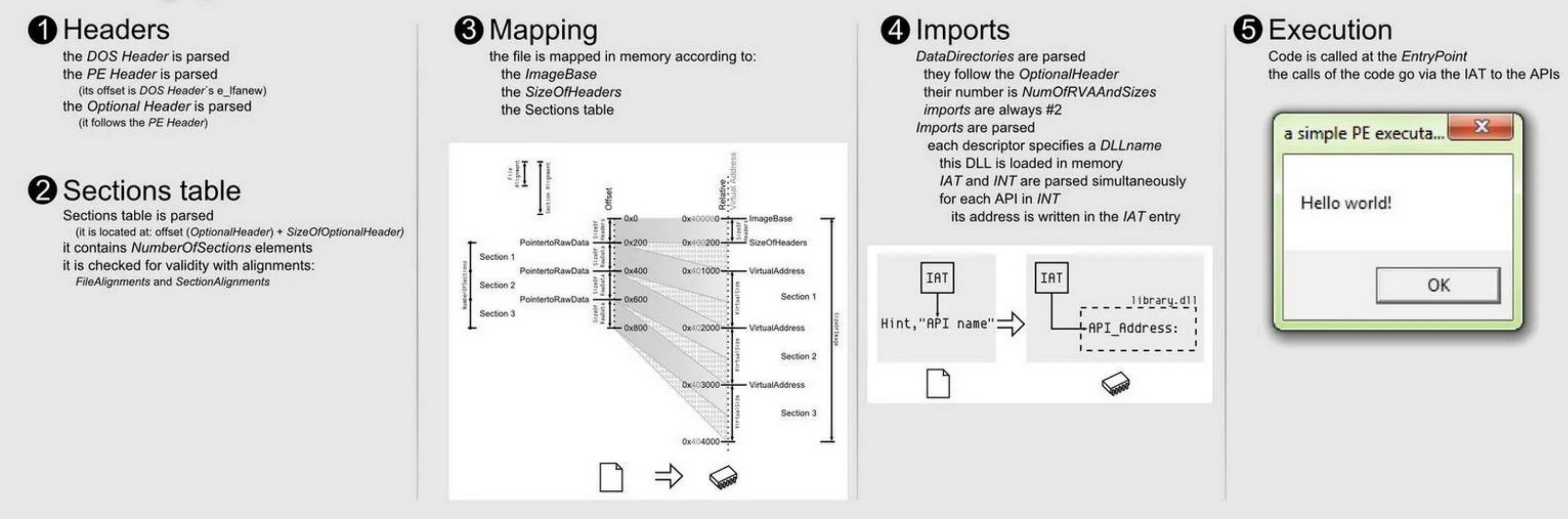
Describes where to find code, initialized data, resources, etc to the loader

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



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

(Thanks Ange Albertini)

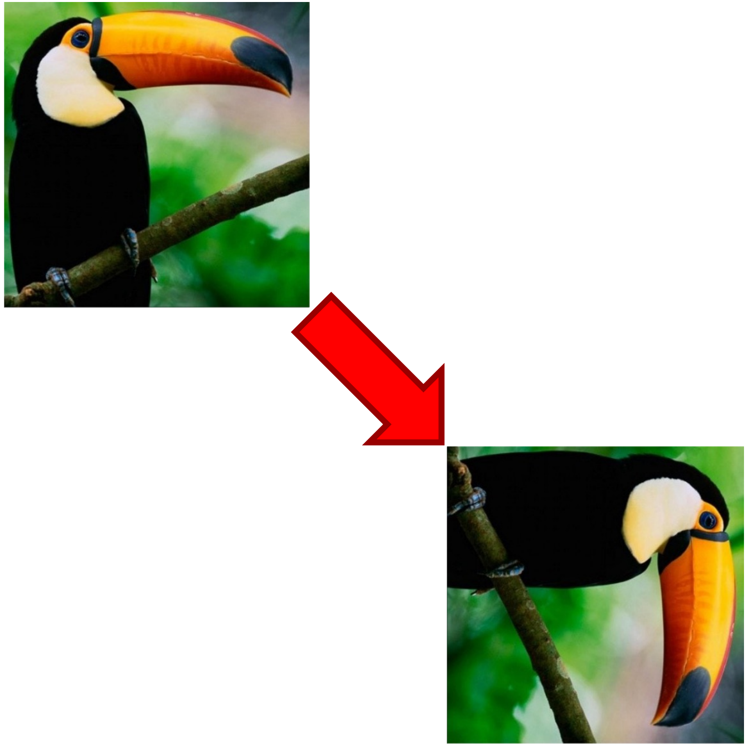
Towards Adversarial EXEmples

Perturb the representation of a file

Keep intact the original functionality

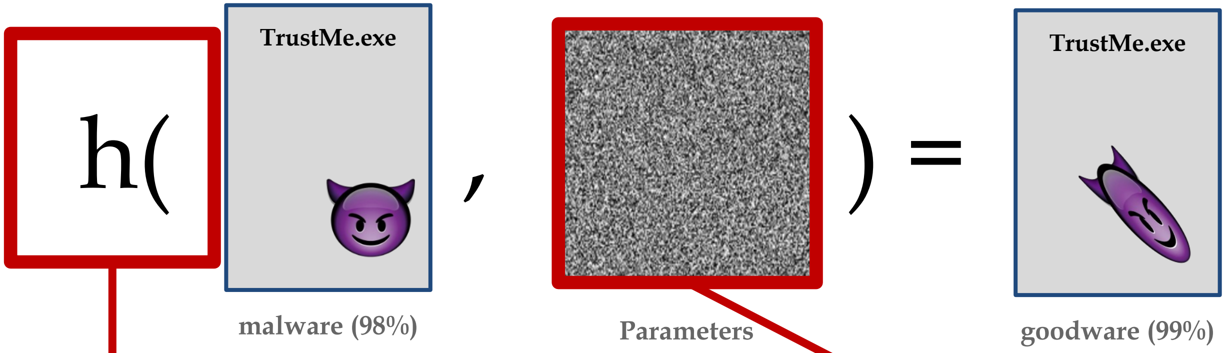
Example: rotation for images

How to bridge the gap?



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

Practical Manipulations

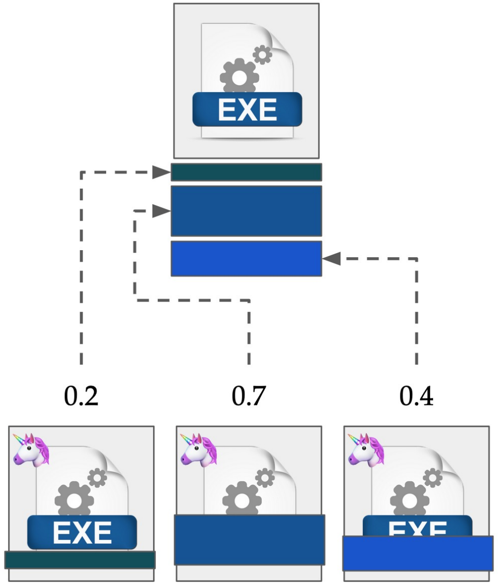


Practical Manipulation function
Alter file representation without destroying the structure and the functionalities and avoid usage of sandboxes

Parameters
Manipulations are parametrized so an optimization algorithm can fine tune them

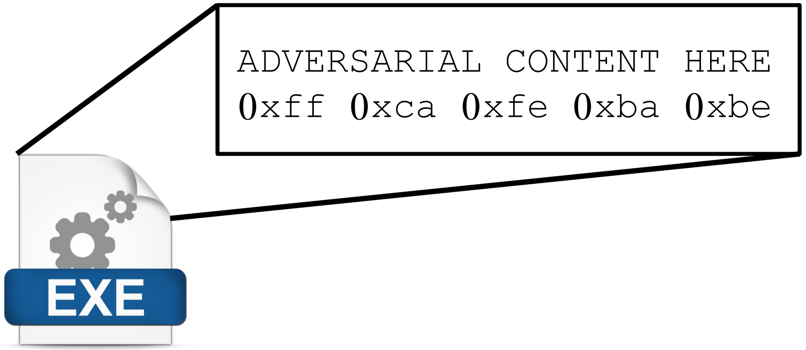
Demetrio et al., *Adversarial EXEmples: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

Structural Manipulations



Injecting content

Alter file structure to include more byte sequences



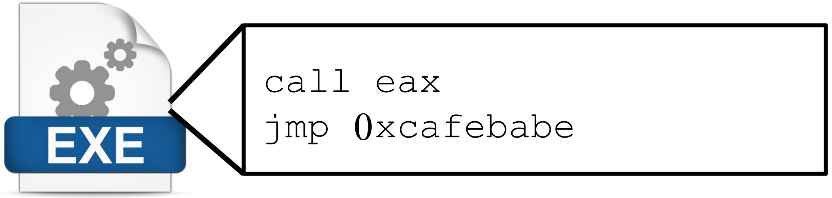
Replacing content

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

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

Behavioral Manipulations

WARNING
more difficult to implement!



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

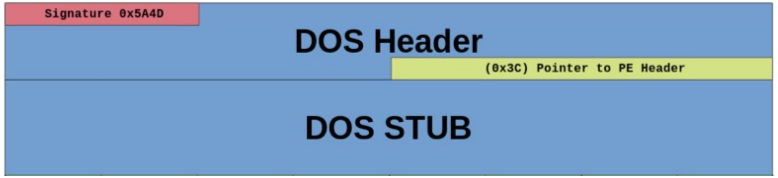
Demetrio et al., *Functionality-preserving Black-box Optimization of Adversarial Windows malware*, IEEE TIFS 2021

DOS header perturbations

The attacker edit as many bytes as they want

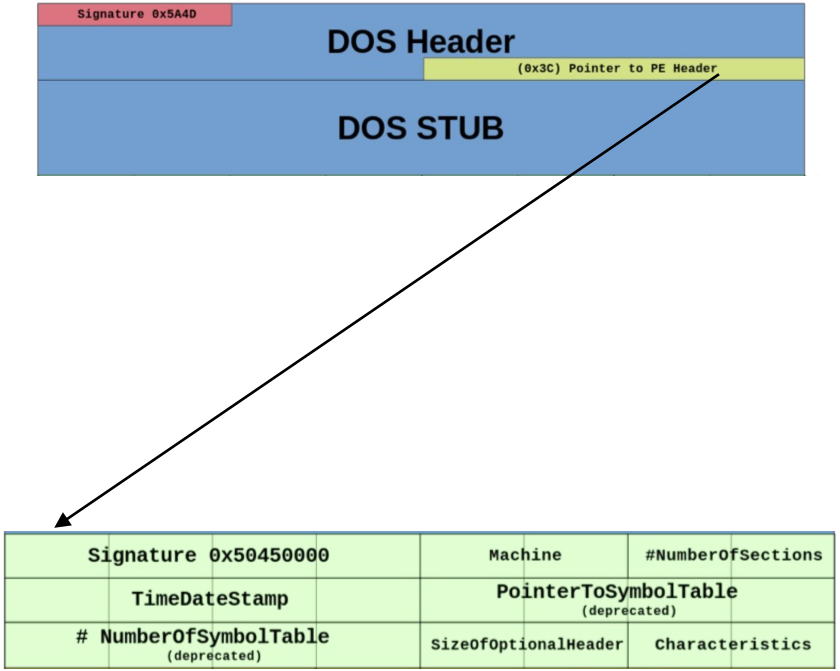
Untouched: magic number MZ and offset to real PE header

Content loaded in memory, not executed



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

DOS header extension



Exploit offset to real header, increment value

Insert arbitrary content between DOS header and PE header

Content loaded into memory, not executed

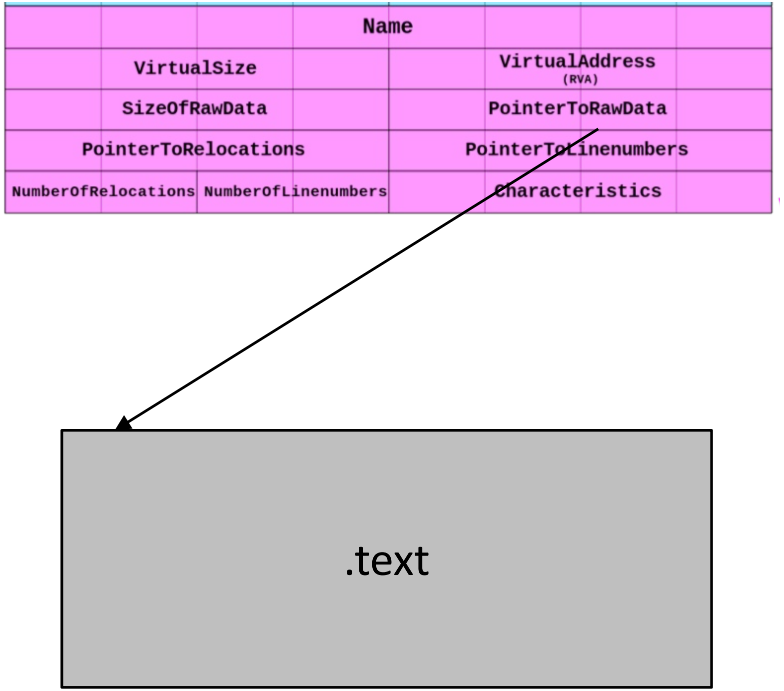
Demetrio et al., *Adversarial EXEmples: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

Content shifting

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



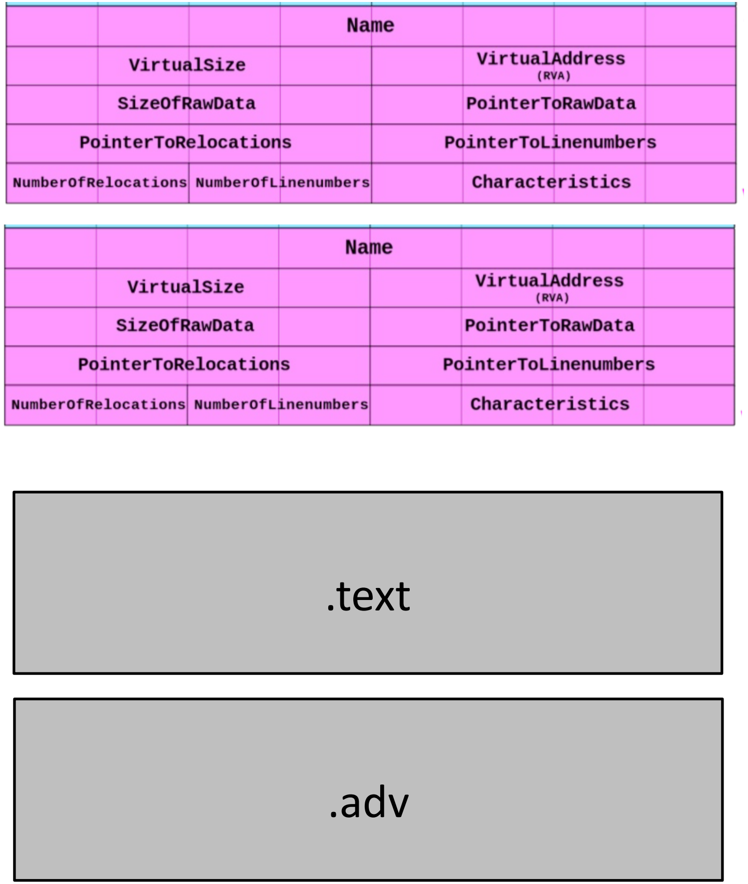
Demetrio et al., *Adversarial EXEmples: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

Section Injection

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



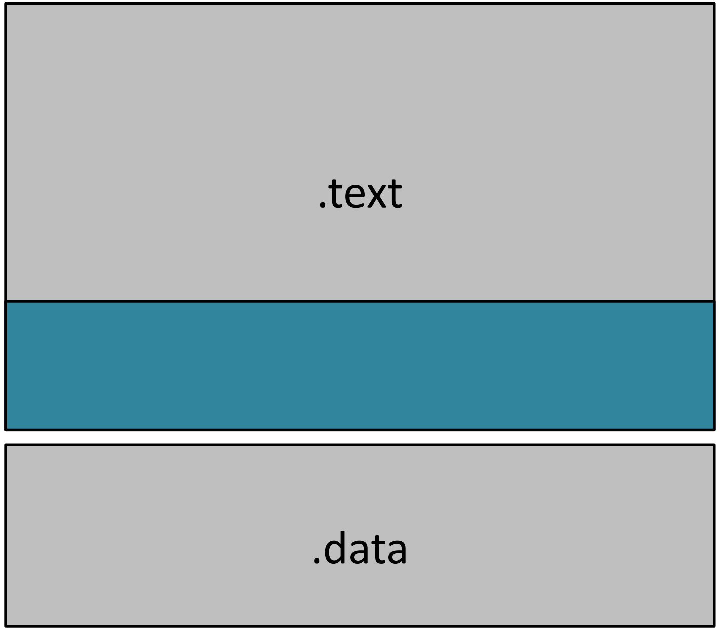
Demetrio et al., *Adversarial EXEmples: a Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection*, ACM TOPS 2021

Slack space

Section content is padded with 0 to keep file alignments

The attacker can rewrite such slack space

Loaded in memory, not executed



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

Padding

Appending content at the end

Most trivial manipulation

Not loaded in memory

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

Optimization Algorithms

Chosing the strategy accordingly

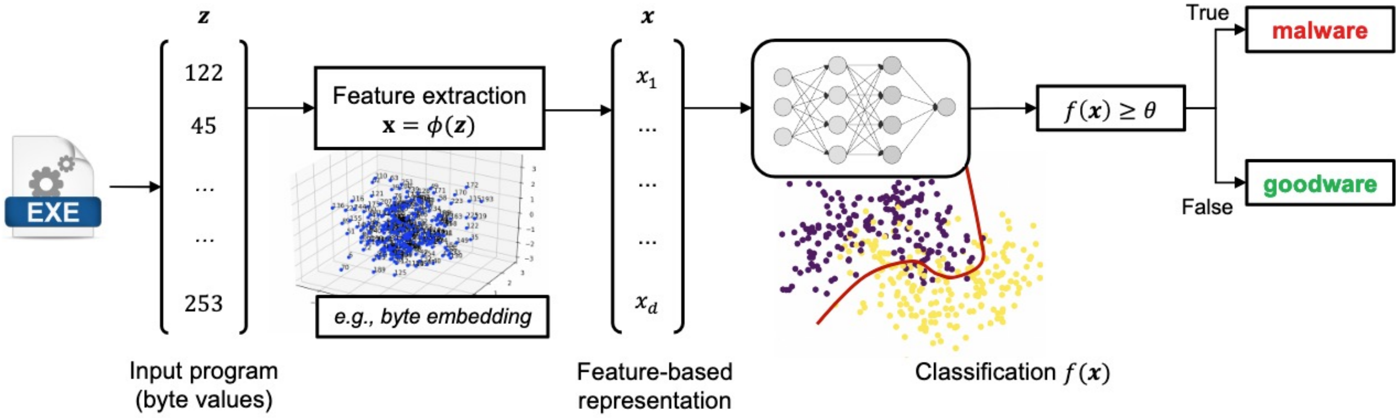
Gradient-based

Own the model
AND
Model is differentiable

Gradient-free

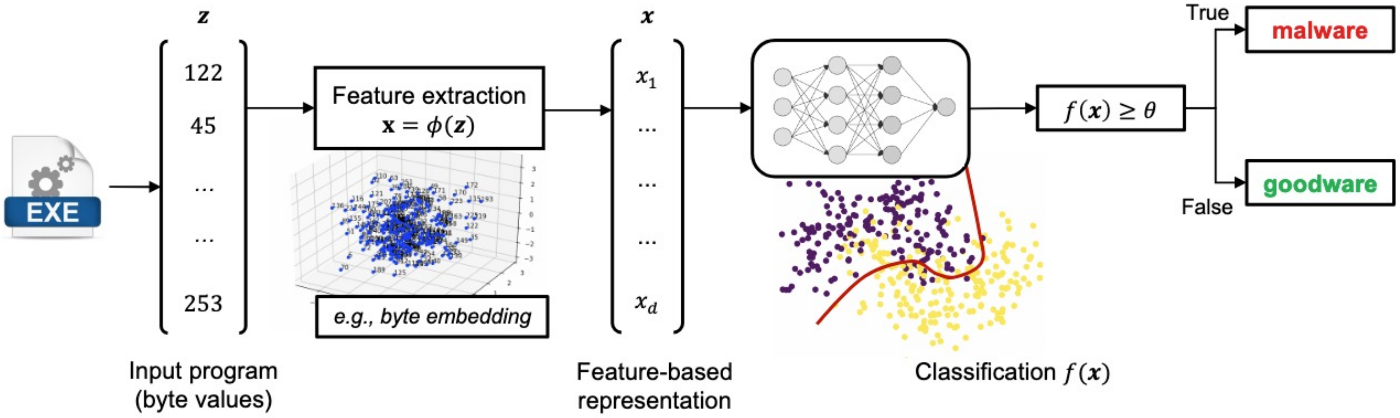
Model not accessible
OR
Model is not differentiable

Gradient-based strategies



Use gradient-descent to compute adversarial examples (as for images)

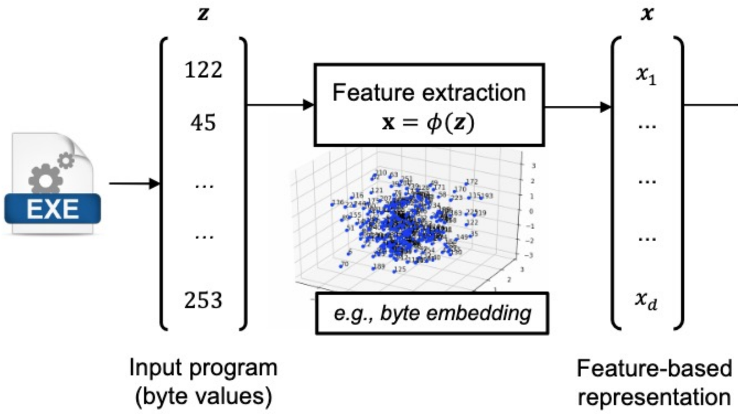
Gradient-based strategies



~~Use gradient-descent to compute adversarial examples (as for images)~~

Bytes do not have a distance metric, a feature extractor is **ALWAYS** needed to compute something meaningful

Embedding for end-to-end networks



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

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

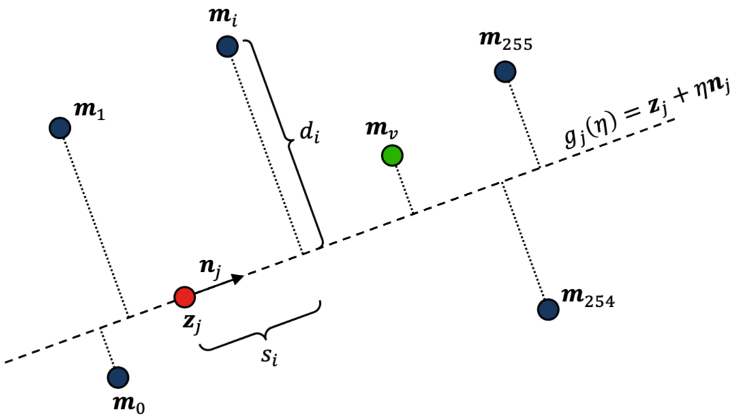
How to propagate gradient information?

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

End-to-end gradient you would like to compute

Non-differentiable manipulations and embedding!

Solution: change the optimizer



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

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

Demetrio, Biggio et al., *Adversarial EXEmples: 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*, EUSIPICO 2018

BGD: Byte Gradient Descent

$\mathbf{E}_i = \hat{\phi}(i), \forall i \in [0, 256]$

$\boldsymbol{t}^{(0)} \in \mathcal{T}$

for i **in** $[0, N-1]$

  $\mathbf{X} \leftarrow \phi(h(\boldsymbol{z}, \boldsymbol{t}^{(i)}))$ // feat. space

  $\mathbf{G} \leftarrow -\nabla_{\mathbf{X}} f(\mathbf{X}) \odot \boldsymbol{m}$

  $\boldsymbol{g} \leftarrow (\| \mathbf{G}_0 \|, \ldots, \| \mathbf{G}_n \|)$

  for k **in** $\text{argsort}\ (\boldsymbol{g})_{0,\ldots,\gamma} \wedge g_k \neq 0$

    for j **in** $[0, \ldots, 255]$

      $\mathbf{S}_{k,j} \leftarrow \mathbf{G}_k^t \cdot (\mathbf{E}_j - \mathbf{X}_k)$

      $\widetilde{\mathbf{X}}_{k,j} \leftarrow \| \mathbf{E}_j - (\mathbf{X}_k + \mathbf{G}_k \mathbf{S}_{k,j}) \|_2$

    $\boldsymbol{t}_k^{(i+1)} \leftarrow \arg\min_{j : \mathbf{S}_{k,j} > 0} \widetilde{\mathbf{X}}_{k,j}$ //input space

$\boldsymbol{t}^{\star} \leftarrow \boldsymbol{t}^{(N)}$

$\boldsymbol{z}^{\star} \leftarrow h(\boldsymbol{z}, \boldsymbol{t}^{\star})$

return $\boldsymbol{z}^{\star}$

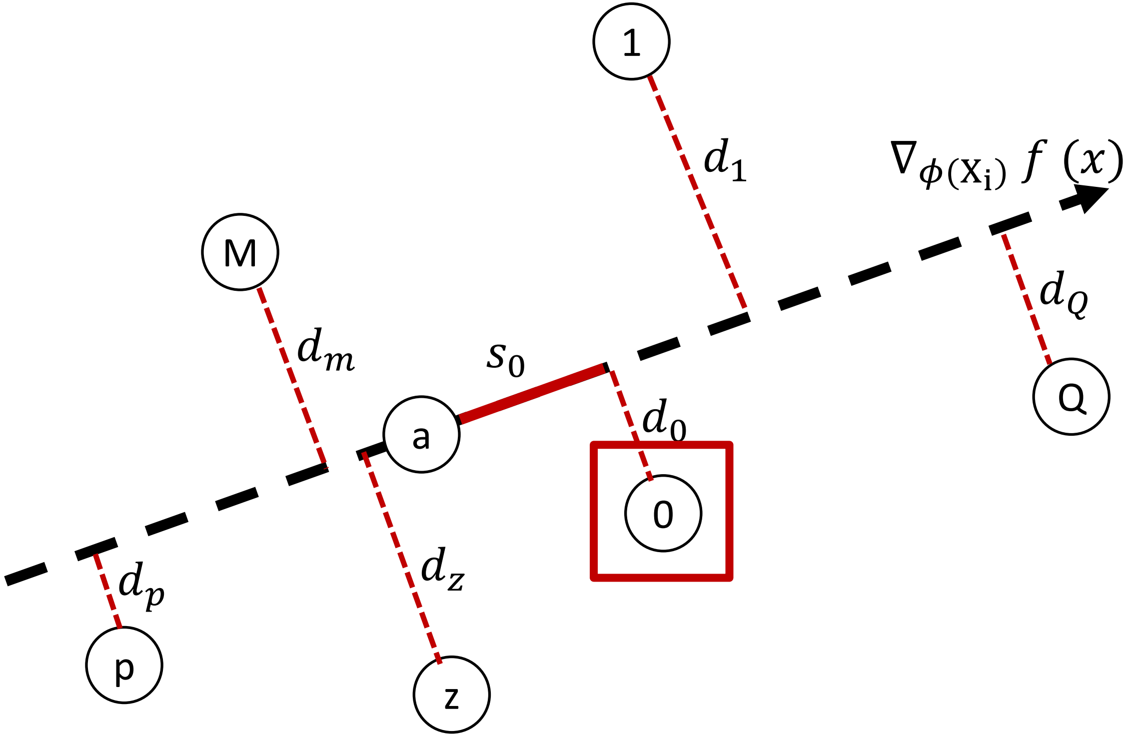
1. Compute gradient in feature space

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 EXEmples: 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*, EUSIPICO 2018

BGD: Byte Gradient Descent (optimization)



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 EXEmples: 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*, EUSIPICO 2018

BGD: Byte Gradient Descent (reconstruction)

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

$$\arg\min_{j:\mathbf{s}_{k,j}>0} \widetilde{\mathbf{X}}_{k,j}$$

Chosing the strategy accordingly

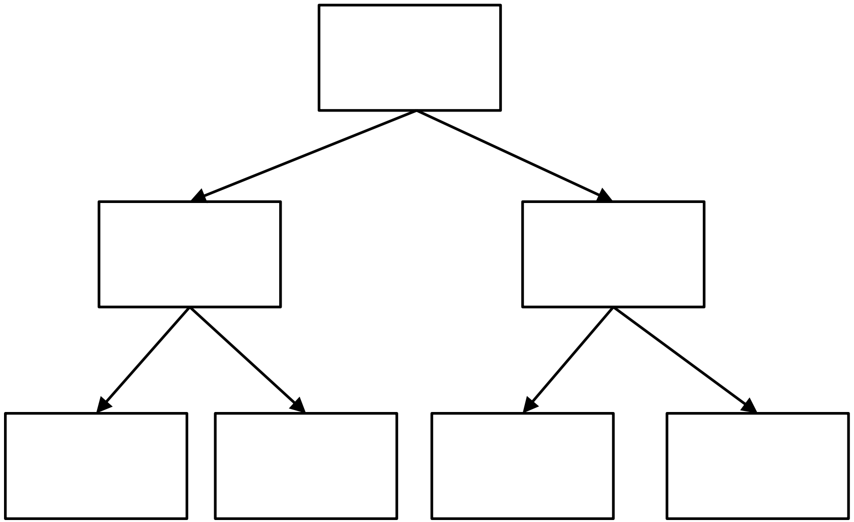
Gradient-based

Own the model
AND
Model is differentiable

Gradient-free

Model not accessible
OR
Model is not differentiable

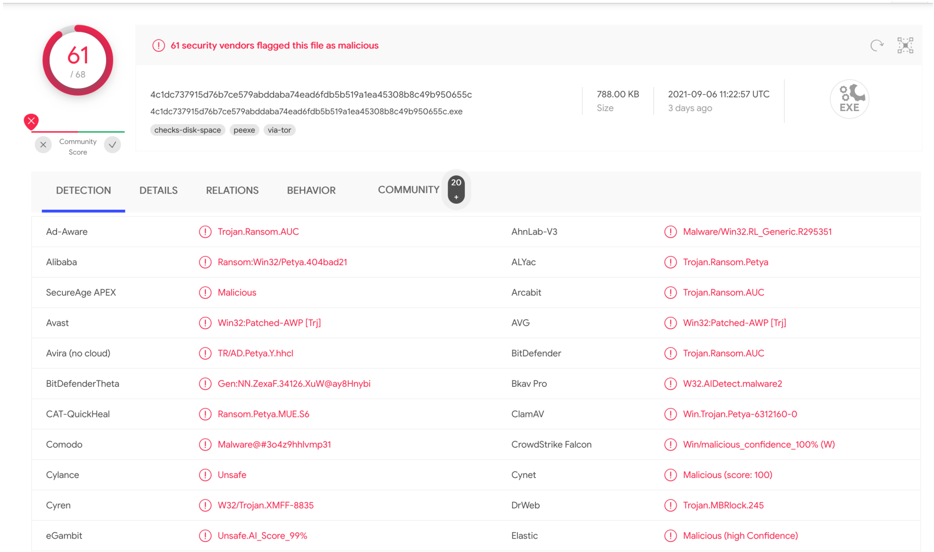
Reality check: robust models are not differentiable



State-of-the-art classifiers use decision trees

No gradients can be computed

Reality check: most models are unavailable

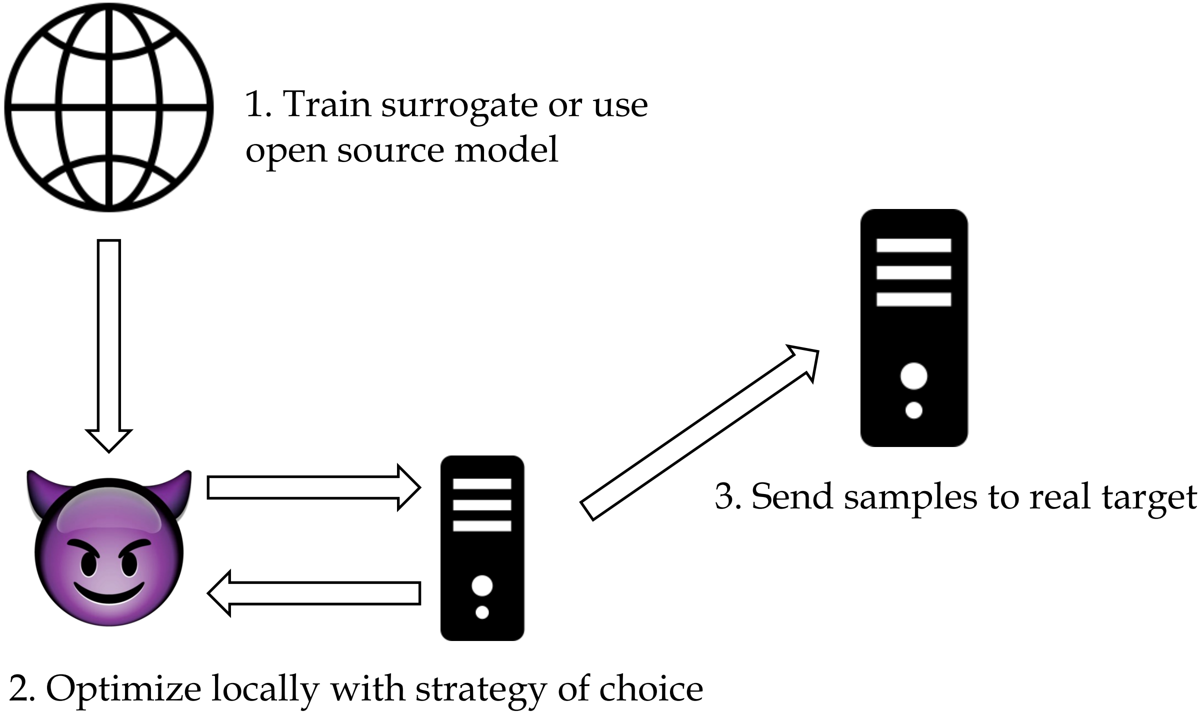


Most models are hosted on private servers

Detection performed in cloud

No gradients can be computed

Transfer attacks



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

Query attacks

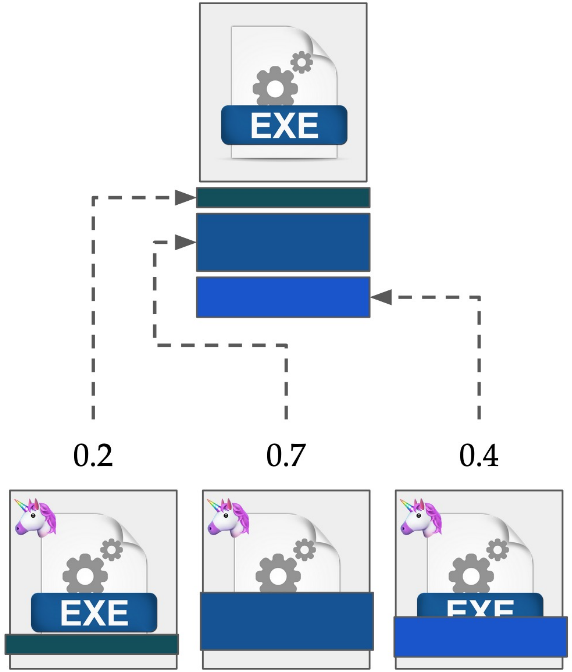
Very slow if optimizer works byte-per-byte

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 explore 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

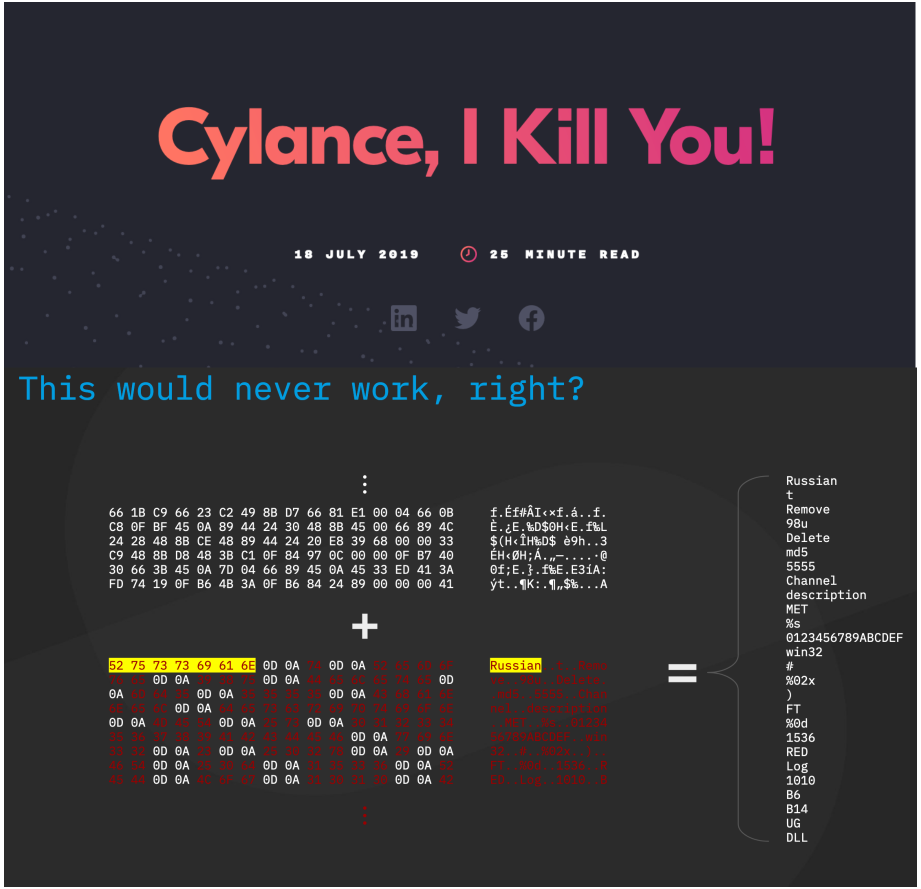
Reversing the code with some tricks, discovered that the model leverages STRINGS

Inject "benign" values

Extract byte sequences from "Rocket League" and include them inside input exacutable

Evasion completed!

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



P.S. they did not, it is still vulnerable on VirusTotal, the write-up is 3 years old now

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

GAMMA against commercial products

	Malware	Random	Sect. Injection
AV1	93.5%	85.5%	30.5%
AV2	85.0%	78.0%	68.0%
AV3	85.0%	46.0%	43.5%
AV4	84.0%	83.5%	63.0%
AV5	83.5%	79.0%	73.0%
AV6	83.5%	82.5%	69.5%
AV7	83.5%	54.5%	52.5%
AV8	76.5%	71.5%	60.5%
AV9	67.0%	54.5%	16.5%

Breaking signatures and patterns
GAMMA (transfer) reduces the perofrmance of commercial products hosted on VirusTotal!

Sneak preview 😎 (1/2)

Genetic algorithms are slow

Optimizing EXEmples with GAMMA requires plenty of time, and it is not easy to control the injected content

Zero-order optimization joins the fight

We are currently working on bringing zero-order optimization inside the world of EXEmples, bending the theory in this non-sensical world without metrics (thank you Marco Rando, Ph.D. student @ MALGA)

$$g_{(G,h)}(x) = \frac{d}{\ell} \sum_{i=1}^{\ell} \frac{f(x + hGe_i) - f(x - hGe_i)}{2h} Ge_i.$$

From theoretical guarantees to EXEmples

First results suggest an improvement in gradient-free optimization attacks against MalConv and GBDT.
Stay tuned for interesting results, and improved optimization algorithms!

Rando et al., *An Optimal Structured Zeroth-order Algorithm for Non-smooth Optimization*, preprint 2023

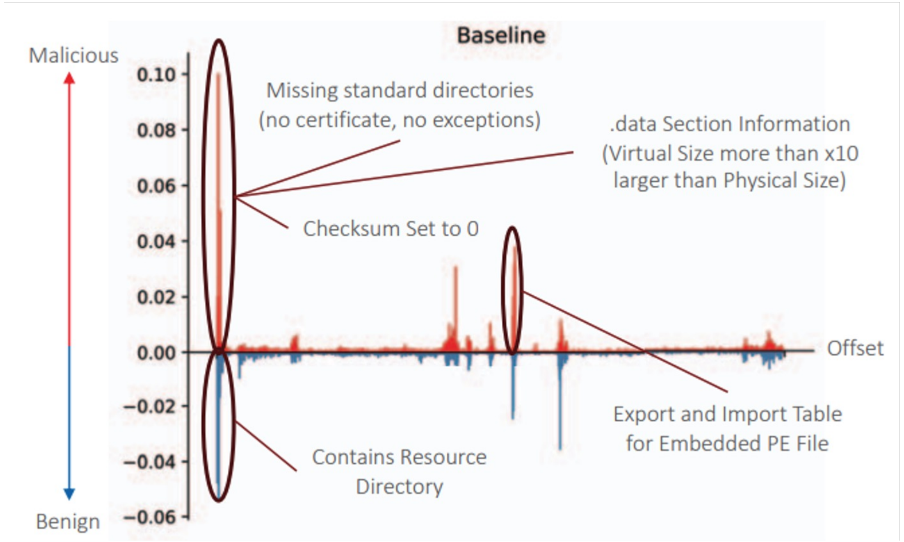
What are these models learning?

Great performances, but why?

Accuracy is high, false positives are low, but most machine learning models are difficult / impossible to inspect

Explainable AI

Train interpretable models (linear, trees) or apply explainablity techniques to demystify decisions

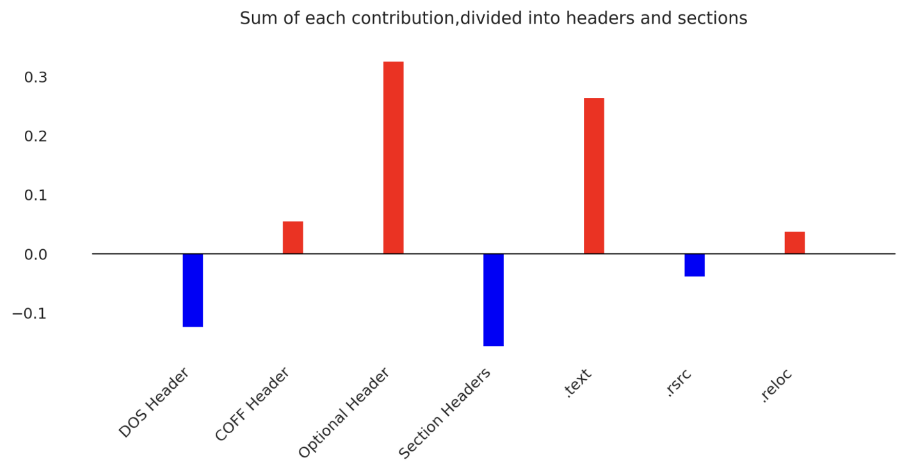


[Coull et al. *Activation analysis of a byte-based deep neural network for malware classification*, SPW 2019]

What are these models learning?

Sometimes, we don’t know!
Our intuition on data is completely different from the correlation learnt by machine learning models

Example: malware detector attributes “legitimate” importance to unused space inside programs!



[Demetrio et al. *Explaining Vulnerabilities of Deep Learning to Adversarial Malware Binaries*, ITASEC 2019]

Sneak preview 😎 (2/2)

What about poisoning?

There are plenty of work on evasive EXEmples, but only few on poisoning of malware detectors

Your PAINT is now an EXEmple

Realistic scenario: attackers embed malicious signatures in regular software, since EVERYBODY trusts VirusTotal

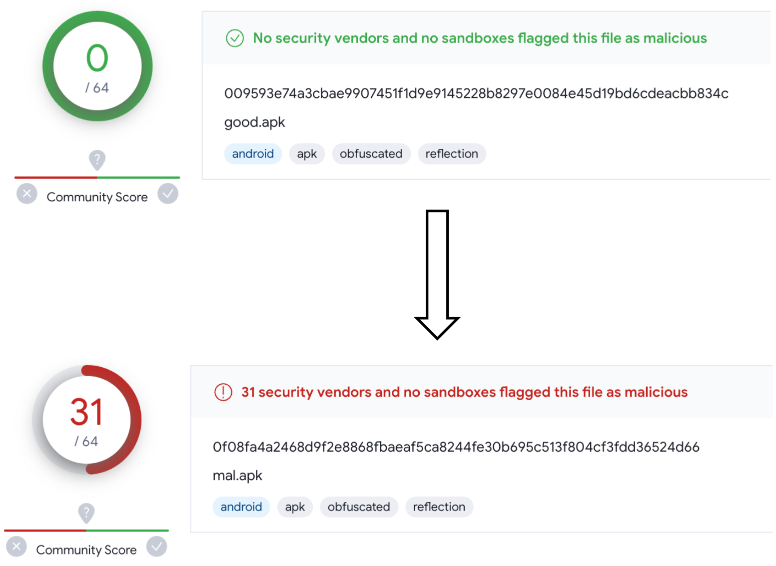
Joint work to show the devastating effect of the exploitation of labelling systems

(thank you Simone Aonzo, Associate Professor @ EURECOM, Han Yufei, Senior Researcher @ INRIA, Tianwei Lan, Ph.D.)

Starting from Android, on EXE is easy

We are working on a more challenging scenario, which is embedding malicious objects inside Android applications

Stay tuned for interesting results!



Rando et al., *An Optimal Structured Zeroth-order Algorithm for Non-smooth Optimization*, preprint 2023

Take-home messages of Part 2

New paradigm: study the format first
Not possible to re-use the same strategies, but first attackers must know how to deal with complex data structures

Adapt already-developed optimization algorithms
Not possible to re-use the same algorithms, since models might be only partially differentiable

Benign content injection rocks
Reduce the search space, faster attacks with effective results

Effectiveness in the real world as well
Evidence show that commercial products might be evaded as well

Are these model learning something?
Yes, but this is not what we expect, and spurious correlations are around the corner

Part 3: How to defend from 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

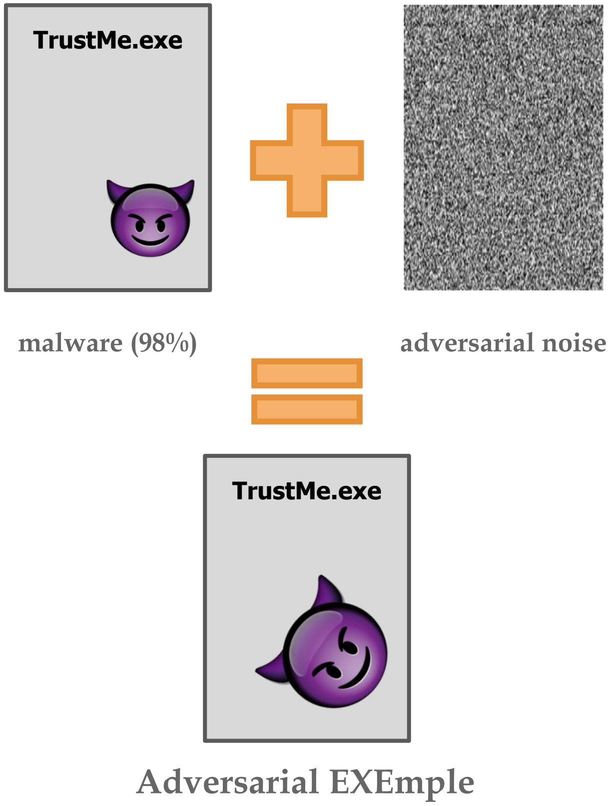
The Windows PE file format is redundant and many components are not used by the operating system at loading time, giving space to the attacker

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



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

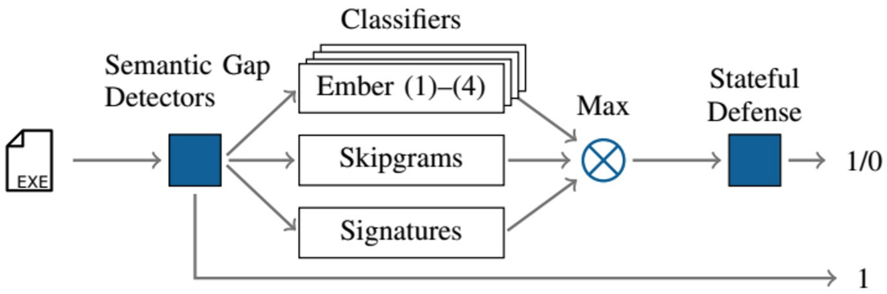
Partially reproducible
There are no pre-trained available, but code is available online
https://github.com/EQuiw/2020-evasion-competition

Unpublished
Still a preprint, never published (proposed for a 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 EXEmples

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

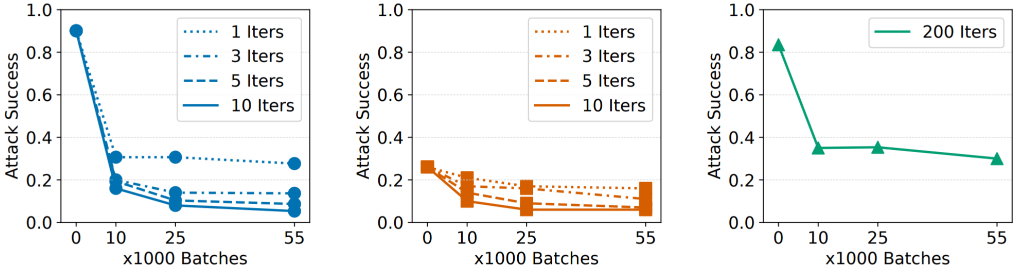
Adversarial Training

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
Samruddhi Pai
Carnegie Mellon University
Weiran Lin
Carnegie Mellon University
Lujo Bauer
Carnegie Mellon University
Michael K. Reiter
Duke University
Mahmood Sharif
Tel Aviv University

Lucas et al. *Adversarial Training for Raw-Binary Malware Classifiers* USENIX 2023

Non-negative Networks

Malicious contributions

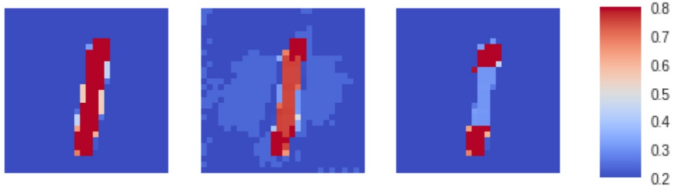
Intuition: classification is based on addition of small malicious triggers, until a threshold is reached

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

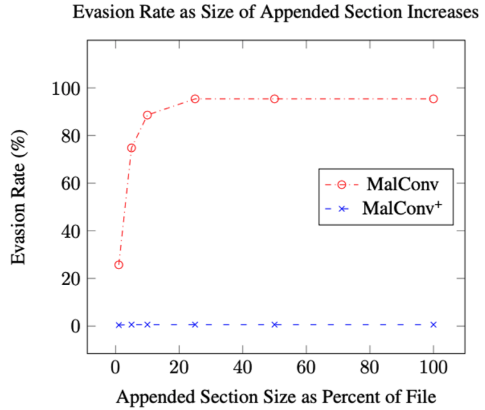
Still a preprint, never published to either conferences, journals or workshops

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,[1] Edward Raff,[1,2] Jared Sylvester,[1,2] Steven Forsyth,[3] Mark McLean[1]
[1]Laboratory for Physical Sciences, [2]Booz Allen Hamilton, [3]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

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

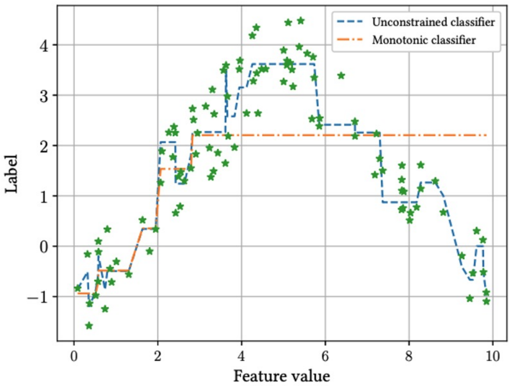
Adversarially Robust Malware Detection Using Monotonic Classification

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



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

Certified Detector


Certified Robustness of Learning-based Static Malware Detectors

Zhuoqun Huang
zhuoqun@unimelb.edu.au
University of Melbourne
Parkville, VIC, Australia

Neil G. Marchant
nmarchant@unimelb.edu.au
University of Melbourne
Parkville, VIC, Australia

Keane Lucas
keanelucas@cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Lujo Bauer
lbauer@cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Olga Ohrimenko
oohrimenko@unimelb.edu.au
University of Melbourne
Parkville, VIC, Australia

Benjamin I. P. Rubinstein
brubinstein@unimelb.edu.au
University of Melbourne
Parkville, VIC, Australia

Formal guarantees

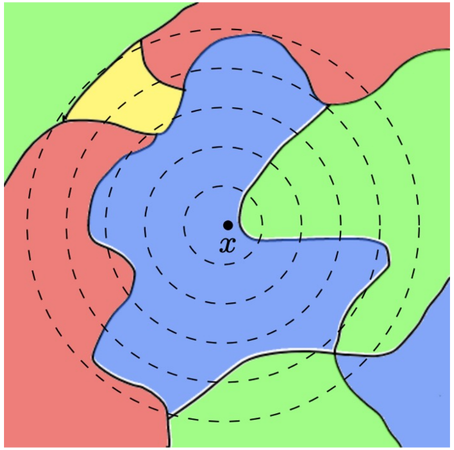
Proofs of non-existence of adversarial examples around input, leveraging edit distance functions.
Formalized on static end-to-end detectors

Not reproducible

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

Published

Accepted at NeurIPS 2023!
Also, many work in this direction are appearing in the state of the art



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

Upcoming innovation in certification

Only padding?

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

Incoming new certification

Joint work on certification for also content-injection attacks that breaks the current methodologies
(combined effort with Daniel Gibert!)

Specific chunking system

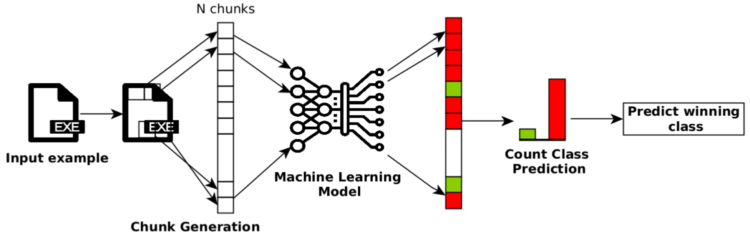
Divide incoming input into chunk according to the format, independently from the size of the file or a fixed number of windows.
Intuition: the adversarial content will always be contained in contiguous blocks, since it must be aligned with a specific entry in the PE file format

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

Detectors of EXEmples

Aiding Avs without removing them

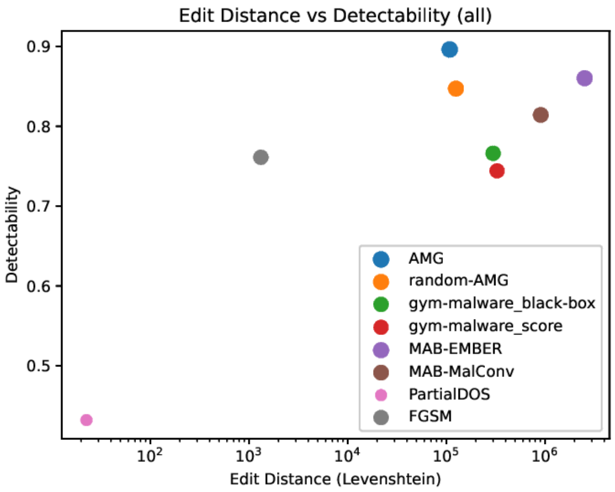
Since it might be difficult to replace a distributed model, we are working to develop a plugin that detects the presence of anomalous EXEmples in test data

Perturbation-spotting

Upcoming work that will show how samples can be discarded if labelled as EXEmples, minimal computational overhead and reduced false positives (thank you Matous Kozak, Ph.D. student @ CTU for this work!)

Trade-off between stealth / effective

The least the EXEmples is modified, the least is detected by scanners: it is a naïve finding, but end-to-end models are more susceptible to "invisible" manipulations than other models!



Take-home messages of Part 3

Research is taking flight
First there was only pre-processing, now we have adversarial training and certification

Need for responsible evaluations
We want to avoid the same history of adversarial defenses on vision models; these new detectors must be evaluated following the practices developed so far

Part 4: Limitations (and future work)

Three big issues with Adversarial EXEmples

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

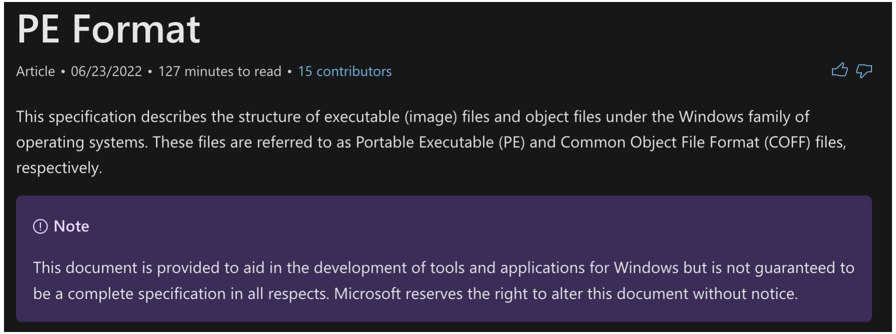
Few available detectors to test
Academic models are either not working
or preliminary, while commercial models
are unavailable

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

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

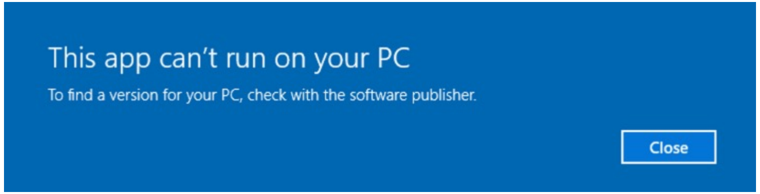

PE Format

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.



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

Mariano Graziano
Cisco Talos
magrazia@cisco.com

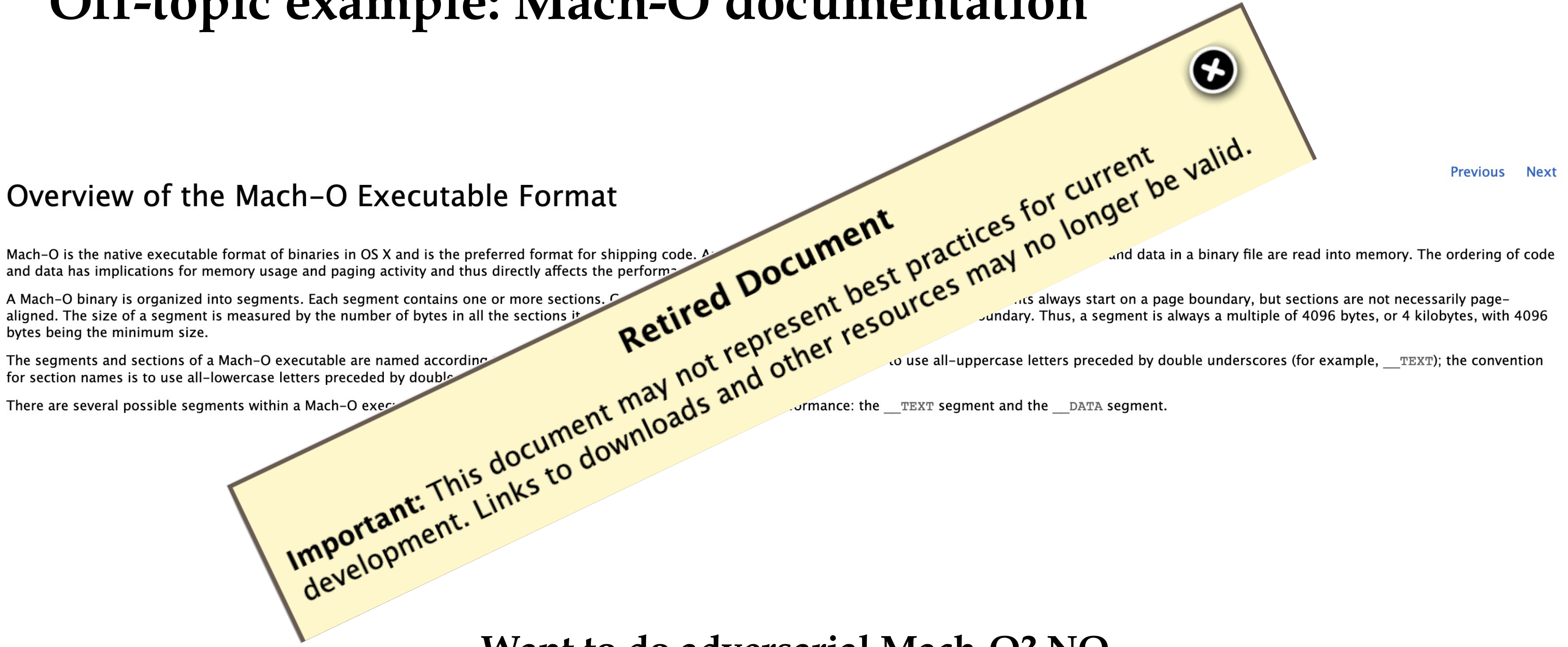
Yanick Fratantonio
Cisco Talos
yfratant@cisco.com

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	✓				✓

[Nisi et al. Lost in the Loader: The Many Faces of the Windows PE File Format. RAID 2021]

Off-topic example: Mach-O documentation



Want to do adversarial Mach-O? NO.

Apple is removing every possible reference to its proprietary program file format, specifics available only through write-ups online

Complex pipeline for debugging manipulations

Dealing with kernel components

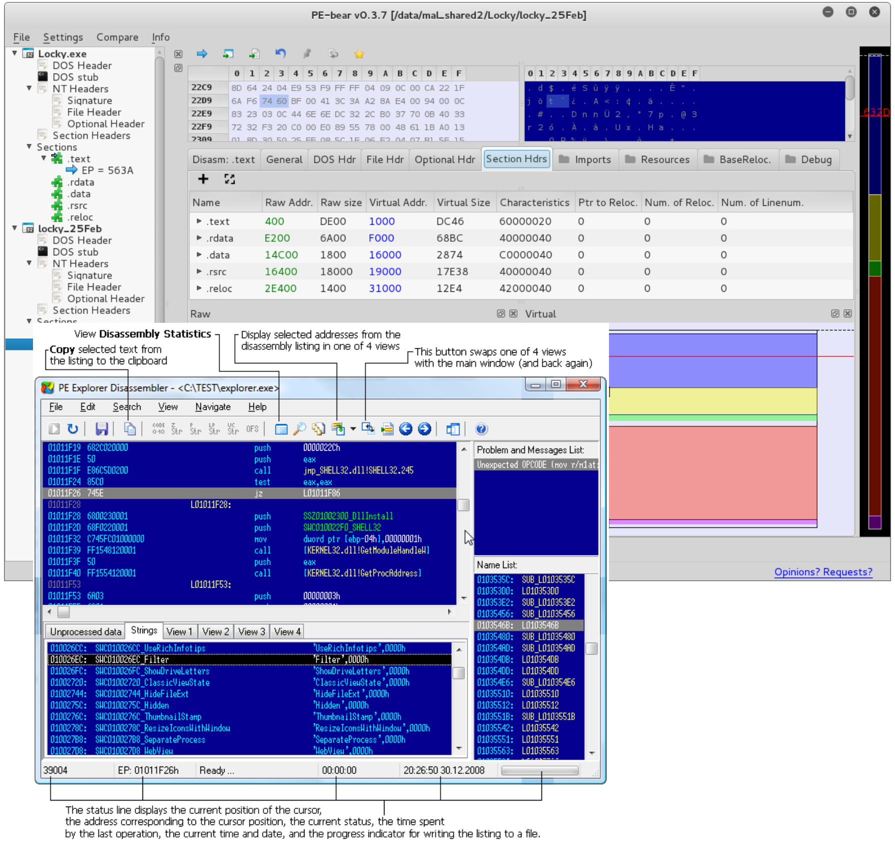
The building blocks of the operating systems are inside the kernel, there is no easy way to connect them to a debugger.
Work-around: manual inspection and tons of wasted hours

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)



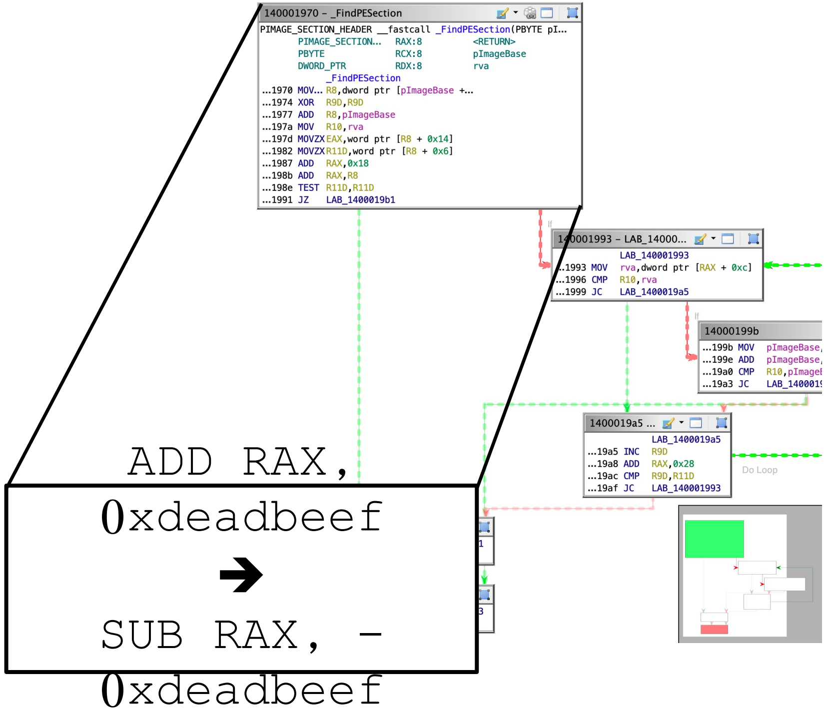
What about attacks against dynamic classifiers?

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…



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

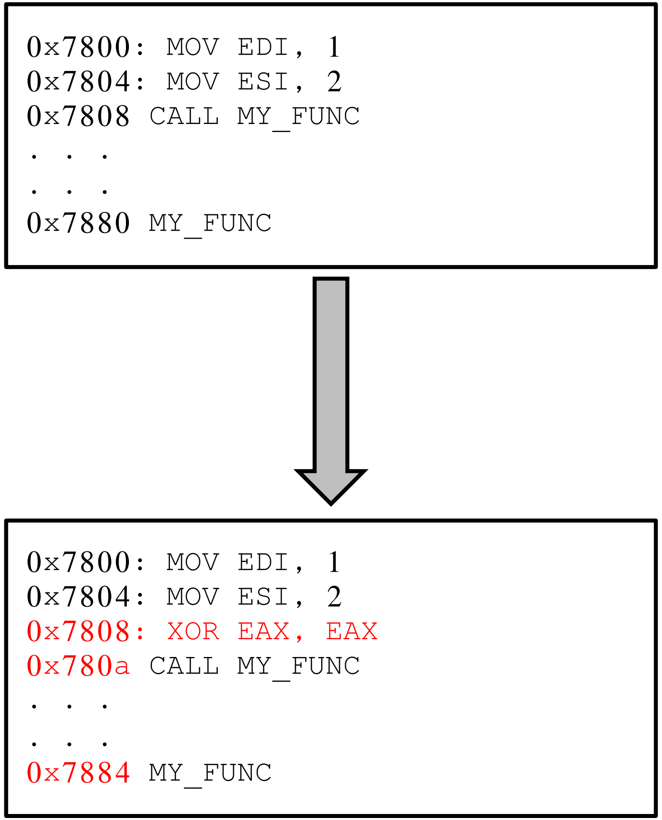
One could create jumps to other code sections… if they are flagged as executable!

Problems with relocations

Program 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 is tons of tools that obfuscate and pack samples, but debugging time levitates from a few to many more hours of human work



Call for action (1/3)

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

Three big issues with Adversarial EXEmples

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

Few available detectors to test
Academic models are either not working
or preliminary, while commercial models
are unavailable

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

Academic classifiers are (kinda) flawed

Academic models are all "unpublished"

Except for MalConv[1], all the other classifiers used in research are published as preprint: EMBER, PEberus, Non-Negative MalConv[2], and many others

No adversarial robustness is considered

Except for PEberus, no model consider adversarial attacks inside their formulation, and they can be easily evaded

Only static detectors

There are no open-source state-of-the-art machine learning models that rely on runtime information of Windows programs[3]

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

This repository contains the defense *PEberus** that got the first place in the Machine Learning Security Evasion Competition 2020, resisting a variety of attacks from independent attackers.

You can find the whitepaper that outlines our defense here:

Elastic Malware Benchmark for Empowering Researchers

The EMBER dataset is a collection of features from PE files that serve as a benchmark dataset for researchers. The EMBER2017 dataset contained features from 1.1 million PE files scanned in or before 2017 and the EMBER2018 dataset contains features from 1 million PE files scanned in or before 2018. This repository makes it easy to reproducibly train the benchmark models, extend the provided feature set, or classify new PE files with the benchmark models.

This paper describes many more details about the dataset: https://arxiv.org/abs/1804.04637

1. MalConv can be evaded by replacing ~60 bytes
2. Non-Negative MalConv has an open-source release that has a terrible ROC

Industry classifiers are locked away

Commercial means Unavailable
All the companies do not share any technical insights about their technologies (of course), most of them can not be tested with a free license, other must be reverse engineered

VirusTotal is the only way
Most of them can be tested using the crowd service VirusTotal, but only old not-updated versions are available

SOPHOS
CYLANCE
CROWDSTRIKE
deep instinct
Windows Defender

These vendors sell machine learning inside their products.

Call for action (2/3)

Develop new state-of-the-art models

We are still using models from 2018, more focus on modelling defenses than creating good and easy-to-use models like MalConv and GBDT

Three big issues with Adversarial EXEmples

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

Few available detectors to test
Academic models are either not working
or preliminary, while commercial models
are unavailable

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

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 malicous content.
Originally created to prevent reverse engineering of legitimate code.

https://resources.infosecinstitute.com/topic/top-13-popular-packers-used-in-malware/

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

Missing the bigger picture

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

<u>Naïve solution: test attacks against deployed commercial products</u>

(which are unavailable, as said before)

Call for action (3/3)

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

Take-home messages of Part 4

Manipulation are hard to craft
Requires patience and hours of work, high risk / high reward scenario

Few classifiers around
We are still using EMBER from 2018 (5 years ago!) with no candidate that performs better, both in terms of accuracy and robustness

Industry are not so concerned (yet)
The focus is mostly shifted towards controlling false positives

Thanks!

Luca Demetrio
luca.demetrio@unige.it
@zangobot

If you know the enemy and know yourself, you need not fear the result of a hundred battles
Sun Tzu, The art of war, 500 BC