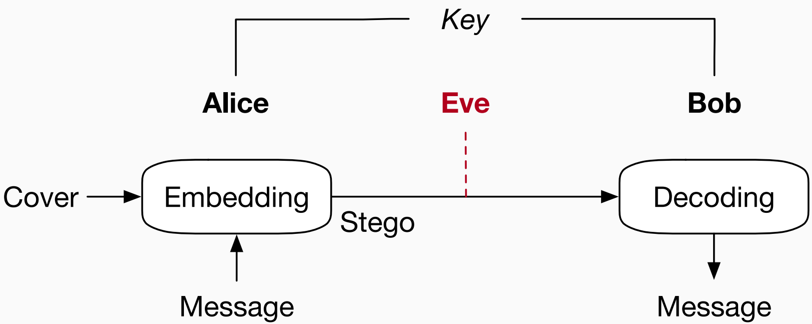
Deep Learning Steganography

to Hide Malware in Web Content

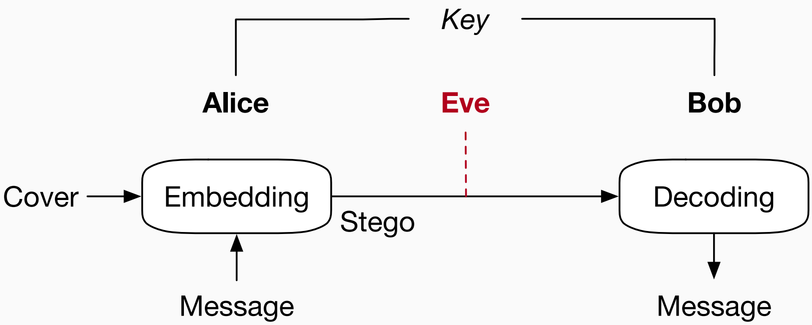
N. Fournaise (Univ. Limoges)
H. Nguyen (ENS Lyon)
F. Recoules (CEA LIST)
T. Taburet (Centrale Lille, Univ. Lille, CNRS)

October 25, 2019

Steganography in a nutshell



Steganography in a nutshell

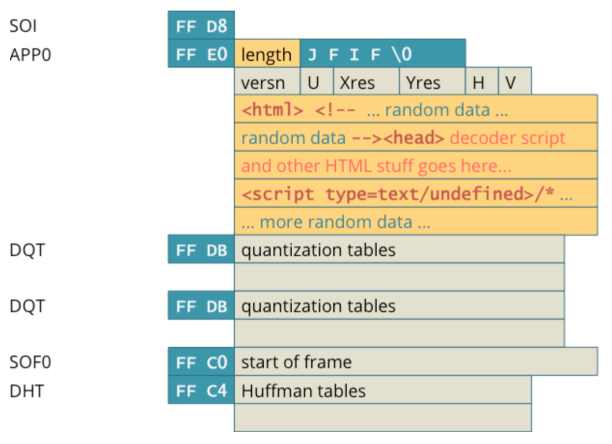


Message can be a malicious code

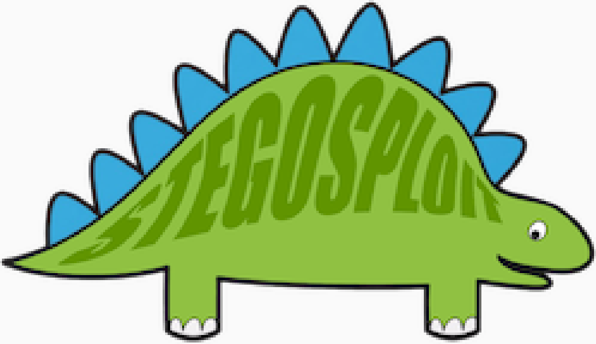
Polyglot file

Polyglot *(Noun)* :

a person who knows and is able to use several languages.

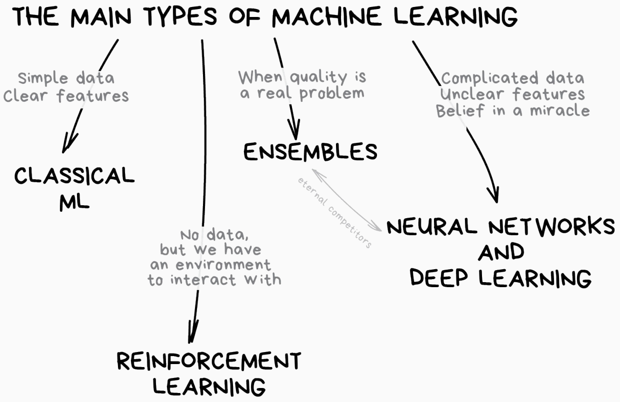


Stegosploit

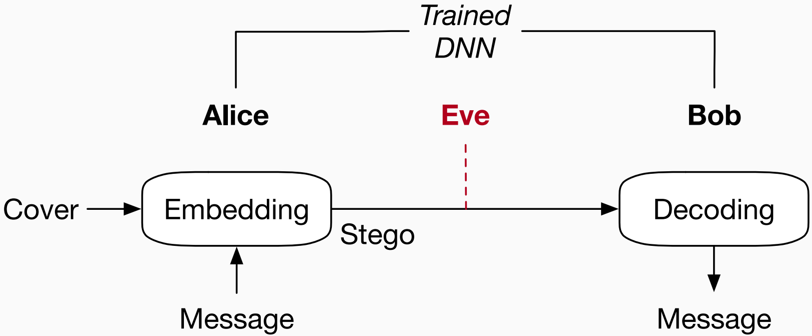


Machine Learning

source: *https://vas3k.com/blog/machine_learning/*



SteganoGAN's case



Huge claims about capacity & security

Can we implement a Machine Learning-based steganographic decoder using web technologies?

Disclaimer

- No GPUs in our laptops
- No Machine-Learning Background

Embedding decoder in a browser

SteganoGAN

- A Steganography algorithm from a Generative Adversarial Network
- Unpublished but public article
- Implementation Available
- Huge claims about capacity and security!

Output

(a) Original image

(b) Basic encoder

(c) Dense encoder

Zoom on basic one

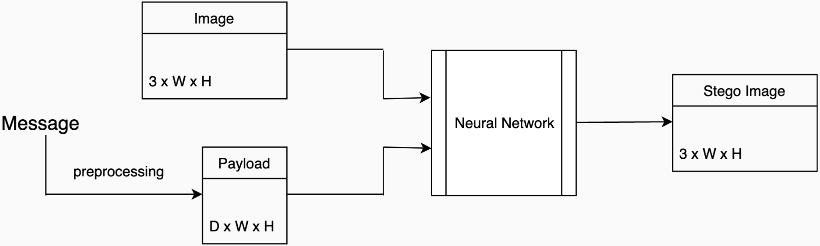
Goal

Adapt the SteganoGAN Decoder part to browser-compatible technologies

One candidate:

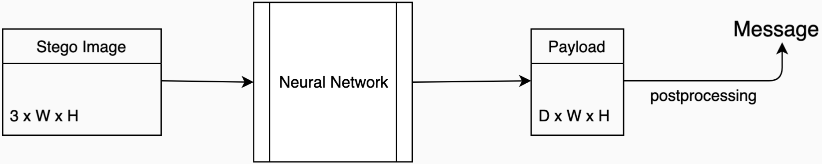
TensorFlow.js

SteganoGAN encoder



Three already trained versions of the Neural Network

SteganoGAN decoder



Essential components

Embed the decoding part *only*

Components:

- Tensor from Image: Python $\Rightarrow$ JavaScript
- Tensor manipulation: PyTorch $\Rightarrow$ TensorFlow
- Neural Network inference: PyTorch $\Rightarrow$ TensorFlow $\Rightarrow$ TensorFlow.js
- Message extraction: Python $\Rightarrow$ JavaScript

Payload construction

Before the encoding part:

RS_ECC(zlib_compression(utf-8_encoding(**Message**))) | 0x00000000

Repeated until no more space in a vector of size D x W x H

Message extraction

After the (decode) neural network inference:

How we improved it

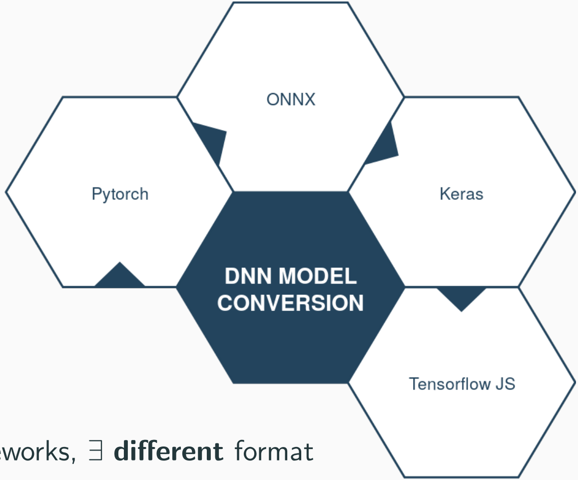
- Define the separators as 4 bytes with a $hamming\_distance < n$
- Use the separators to deduce message length
- From message length, compute the most common bytes for every message character
    - zlib optional
    - Reed-Solomon ECC no longer needed
    - Much faster decoding for large images

How we improved it

- Define the separators as 4 bytes with a $hamming\_distance < n$
- Use the separators to deduce message length
- From message length, compute the most common bytes for every message character
  - zlib optional
  - Reed-Solomon ECC no longer needed
  - Much faster decoding for large images

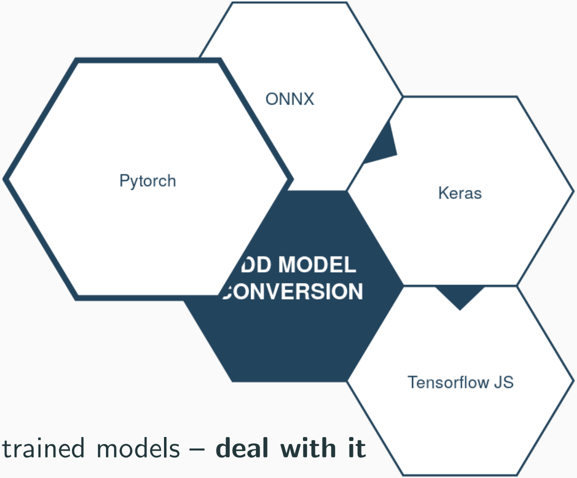
Easier to translate to JS

From Pytorch to Tensorflow JS...



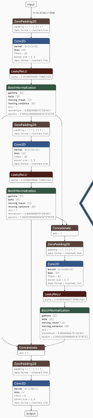
$\forall$ frameworks, $\exists$ **different** format

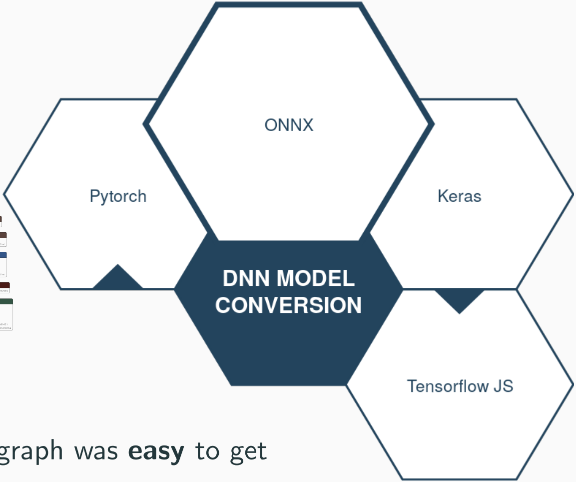
From Pytorch to Tensorflow JS...



Already trained models – **deal with it**

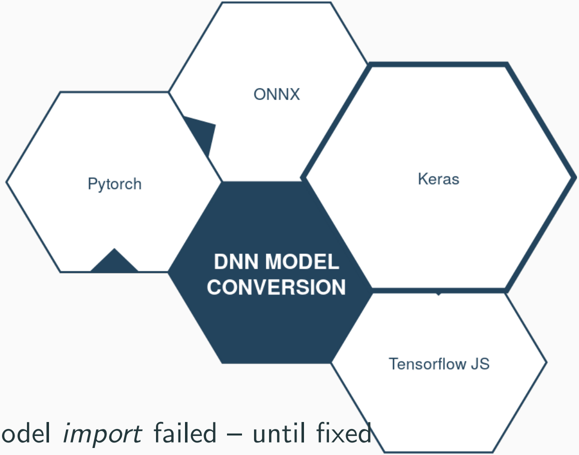
From Pytorch to Tensorflow JS...





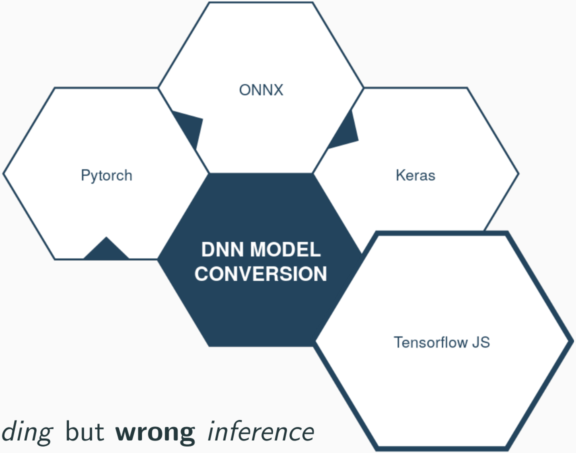
ONNX graph was **easy** to get

From Pytorch to Tensorflow JS...



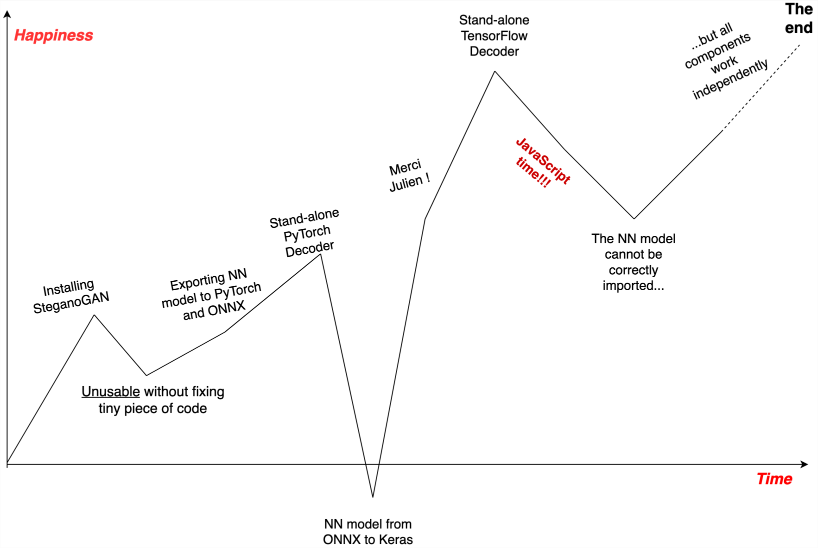
Keras model *import* failed – until fixed

From Pytorch to Tensorflow JS...



Fixed *loading* but **wrong** *inference*

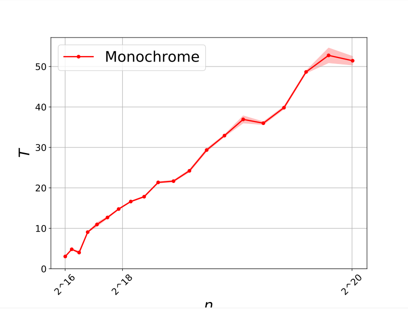
Our journey


Happiness
Installing SteganoGAN
Unusable without fixing tiny piece of code
Exporting NN model to PyTorch and ONNX
Stand-alone PyTorch Decoder
NN model from ONNX to Keras
Merci Julien !
Stand-alone TensorFlow Decoder
JavaScript time!!!
The NN model cannot be correctly imported...
...but all components work independently
The end
Time

Benchmarking SteganoGAN

Benchmark on CPU back-end (1/2)

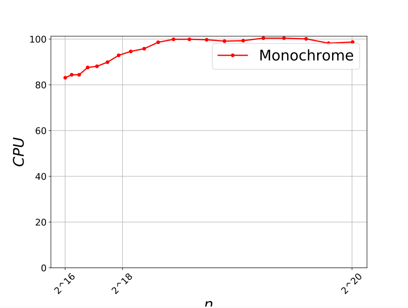
Decoding time as a function of image sizes



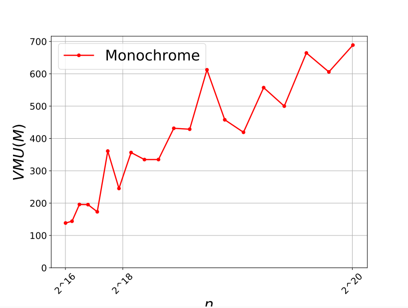
Stealthy decoding CPU implementation implies small images.

Benchmark on CPU back-end (2/2)

CPU usage as a function of image sizes



VRM usage as a function of image sizes



The footprint of CPU/VRM is not sneaky.

Size of javascript exploits

		mean size (B)	min size (B)	max size (B)
without compression (total: 179 exploits)		427.4	42	3871
`uglifyjs` (42 exploits)	compressed	263.8	40	2025
	not compressed	357.5	42	2839

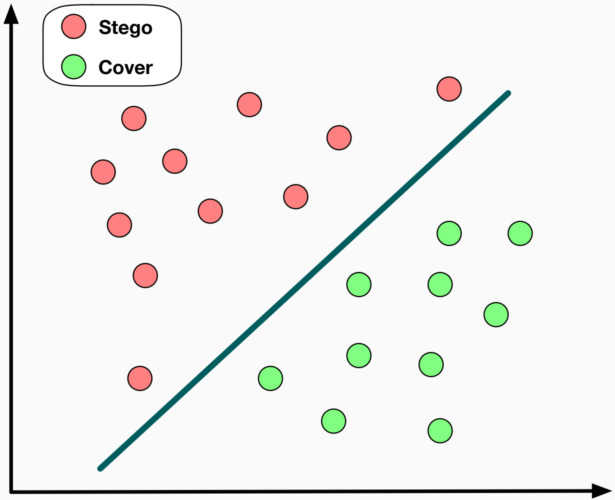
- average compression gain: 26.2%
- average compressed files size: 315.4 B

Steganalysis (1/2): Hand crafted features sets

	Dimensions	Domain
SRM	34671	Spatial
SRMQ1	12753	Spatial
maxSRM	34671	Spatial
DCTR	8000	JPEG
GFR	17000	JPEG
... + 20 others	...	...

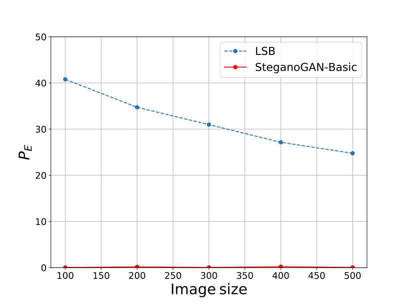
Available on *dde.binghamton.edu*

Steganalysis (2/2): (Low complexity) Linear Classifier

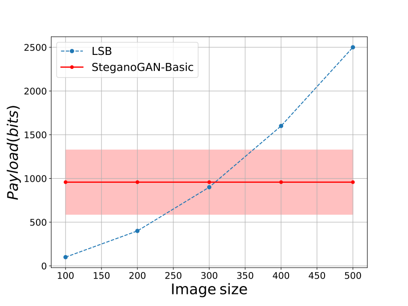


Steganalysis using trained adversary (SRMQ1 features set)

PE : Probability of error



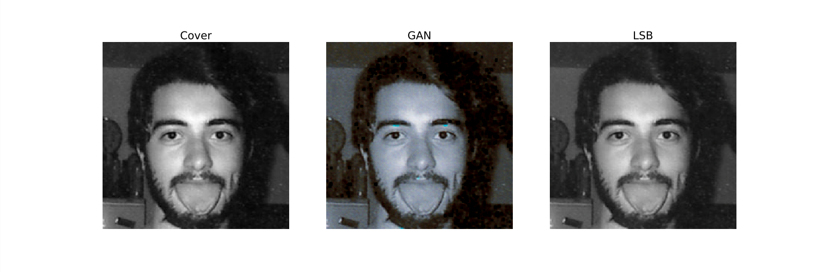
Message size

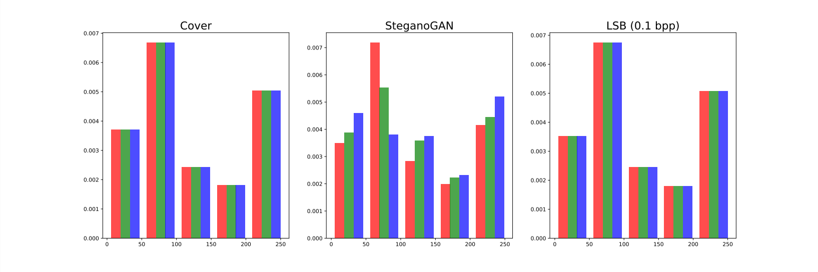


$$P_E = \frac{1}{2}(F_P + M_D)$$

Bonuses

SteganoGAN's weakness (one of them)





Stegosploit countermeasure (WebMaster POV)

If users are allow JPEG/PNG/... to upload file on the website :

- Place these assets on a separate domain.
- Rewrite the JPEG header to ensure no code is sneaked in there and remove all JPEG comments.
- Refuse requests whose type is "script" and source has a MIME type that starts match an image format

Literature about Steganography/Steganalysis

1. Traditional algorithms
    - Spatial-domain
    - Transform-domain
2. Deep learning-based algorithms

Discussion

- How to achieve a balance between security and capacity?
- How to improve the quality of steganographic image from the ML based large capacity steganography algorithm?
- How to consider complexity?

Conclusion

Already done

- one error away from full **POC**
- some clues about (SteganoGAN)
  - bad performance
  - bad security

Future works

- train a proper model for TensorFlow JS
- is steganography relevant for exploits?