PhishGNN: A Phishing Website Detection Framework using Graph Neural Networks

Tristan BILOT
Badis HAMMI
Grégoire GEIS

EPITA Systems and Security Laboratory (LSE)
EPITA Engineering School

19th International Conference on Security and Cryptography
Secrypt 2022

Plan

Plan

Context

Phishing Attacks in Recent Years

- The number of phishing websites between January 2016 and January 2021 increased tremendously
- We need a reliable way to prevent this growth of attacks

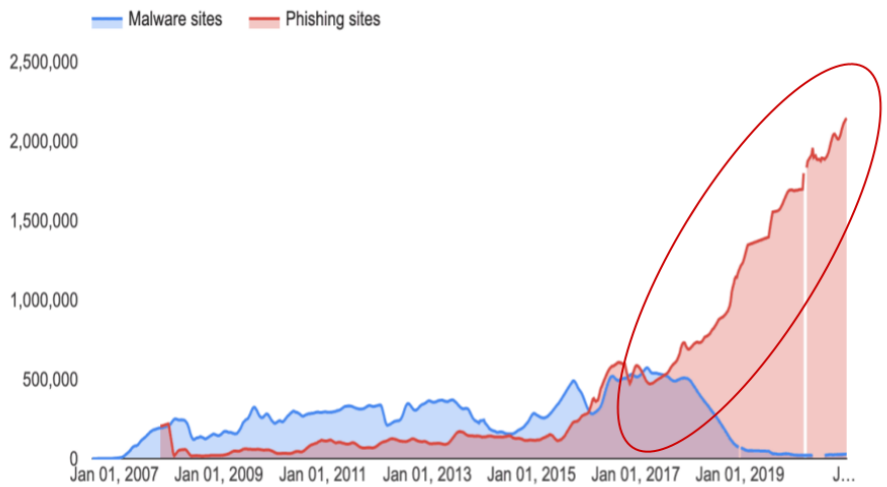


Figure – Number of phishing websites observed between 2007 and 2021. Source : Google Safe Browsing

Context

What Already Exists in Phishing Detection ?

- URL blacklists
- Heuristics
- Machine Learning-based approaches
  - Naive Bayes
  - Support Vector Machine (SVM)
  - Logistic Regression
  - ...
- Deep Learning-based approaches
  - Multi-Layer Perceptrons (MLP)
  - Convolutional Neural Networks (CNN)
  - Generative Adversarial Networks (GAN)
  - Recurrent Neural Networks (RNN)
  - Long Short-Term Memory (LSTM)
  - ...

Research problem

Website Hyperlink Structure

- Most existing techniques do not leverage the internal hyperlink structure of phishing websites for its detection
- The only known implementations are based on hand-crafted graph features
- These implementations do not leverage a Deep Learning model to let it learn the features itself
- Graph features learned by such a model along with other traditional features could lead to better classification performance

Contributions

Contributions

- A framework based on Graph Neural Networks for phishing website detection
  - Our solution leverages the hyperlink structure thanks to Graph Deep Learning, along with many other hand-crafted features learned with traditional Machine Learning
- A Rust crawler for extracting the graph structure of websites has been made open-source [a]
- The dataset we built and used during this study is also publicly available

a. `https://github.com/TristanBilot/phishGNN`

Plan

Related works

Tan et al. (2020) [1]

- Propose a graph-based detection system where hand-crafted features are extracted from the hyperlink structure of the webpage
- Achieve 97.8% accuracy using a C4.5 classifier on these features
- The authors do not leverage the Deep Learning to let the model learn by itself the most useful features to differentiate benign and phishing cases
- A dataset of only 1000 samples is used (500 benign, 500 phishing)

Ouyang and Zhang (2021) [2]

- Only study applying GNNs to phishing detection
- A graph is built from the HTML DOM and a GNN is fed with this graph to perform predictions with a 93% accuracy
- Relies solely on the HTML content
- Does not leverage other features which lead to good results in other studies

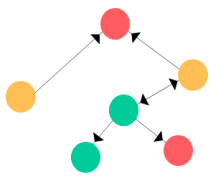
Plan

Contribution

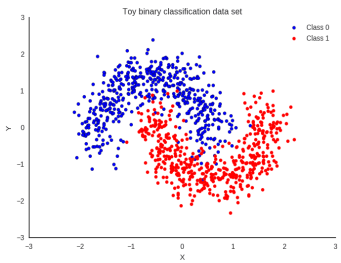
1 → **2** → **3**

```
<div align="center">

<a href="http://deondon.com/">Home</a> -
<a href="http://deondon.com/tribe/“ target="_blank">Subscribe</a> -
<a href="http://deondon.com/contact/ “ target="_blank">Contact</a> -
<a href="http://deondon.com/followme/“>Facebook</a> -
<a href="http://deondon.com/followme/“>Instagram</a> -
<a href="http://deondon.com/followme/“>Twitter</a> -
<a href="http://deondon.com/followme/“>LinkedIn</a><br>
&copy; Copyright 2016 by Deon Don.<br>

 </div>
```

Extract the architectural **structure** of a web page along with some **features**.



Create a **graph** based on every link crawled on the page.



Build an algorithm which will predict if the page is **Phishing** or **Benign**.

Feature extraction

- A vector of size 25 features is extracted from every crawled URL
- 3 sets of features : lexical, content and domain features
- <a>, <form>and <iframe>tags are used by the crawler to build the graph
- The graph is stored as a COO-format matrix of shape $2x|\mathscr{E}|$
- Thus each graph is stored using only $\mathscr{O}(|\mathscr{E}|)$ memory space

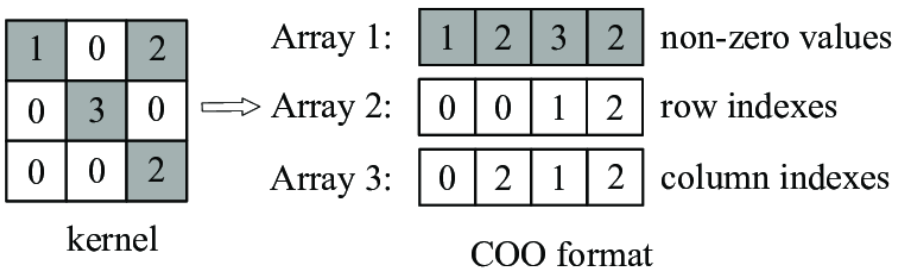


Figure – COOrdinates format used to store graph matrices.

Feature extraction

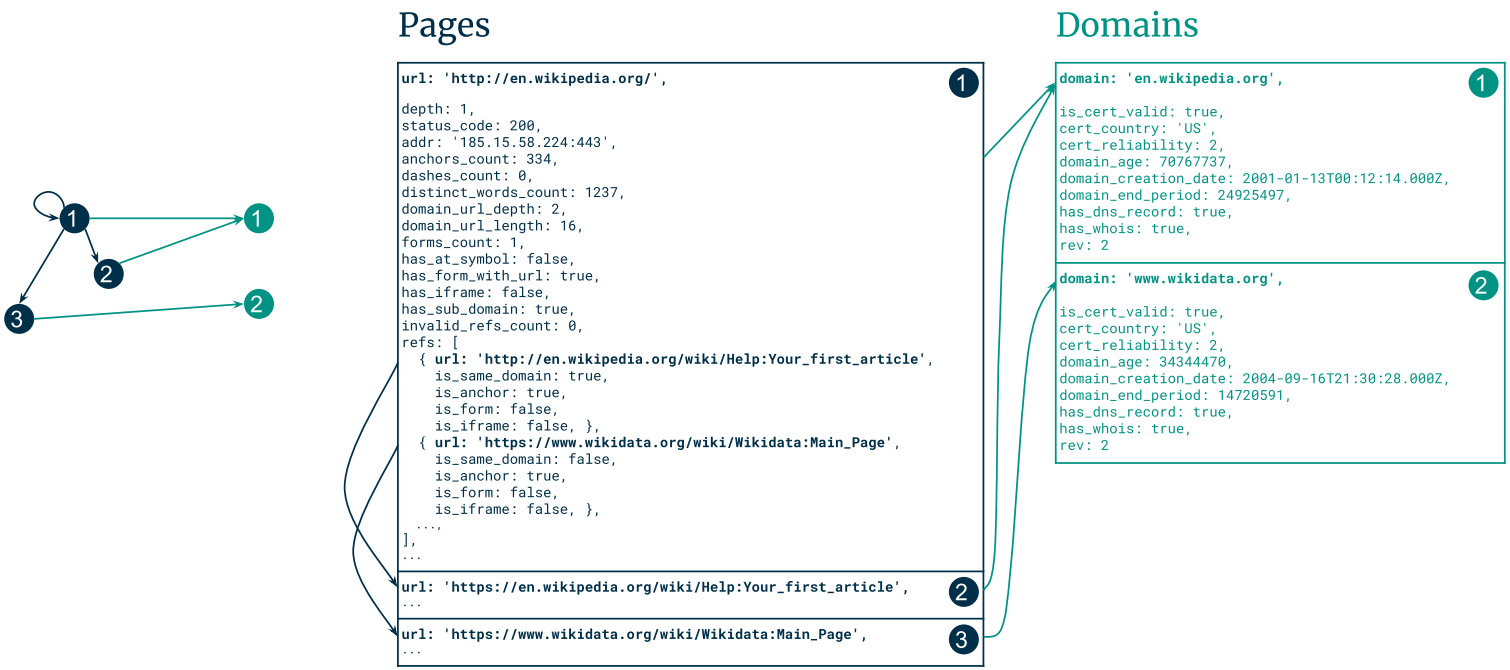
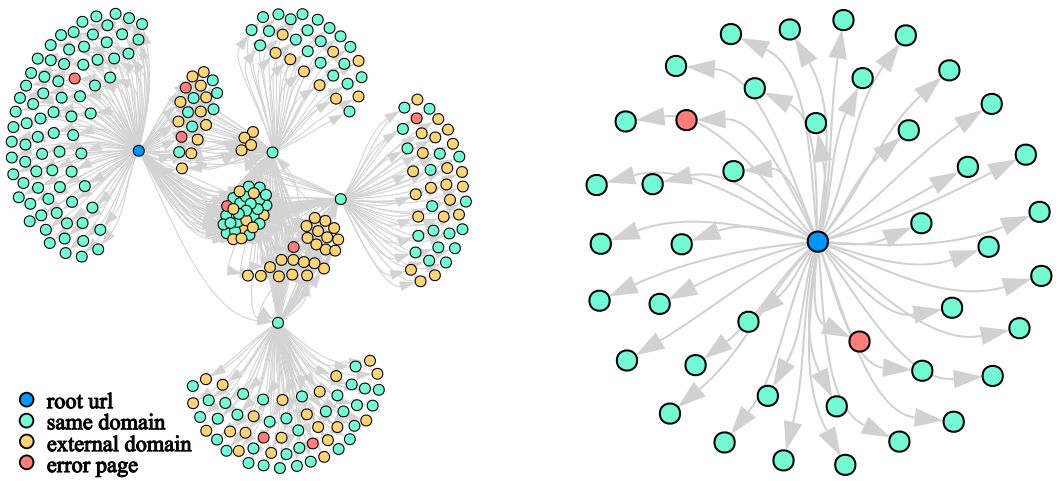


Figure – Extracted features for every crawled URL.

Graph visualization



(a) (b)

Graph representation of two websites after crawling with depth=1. Graph on the left contains multiple children URLs already crawled in previous iterations so their children are inserted in the graph as nodes of depth 2. Graph on the right contains children URLs never crawled before.

Graph visualization

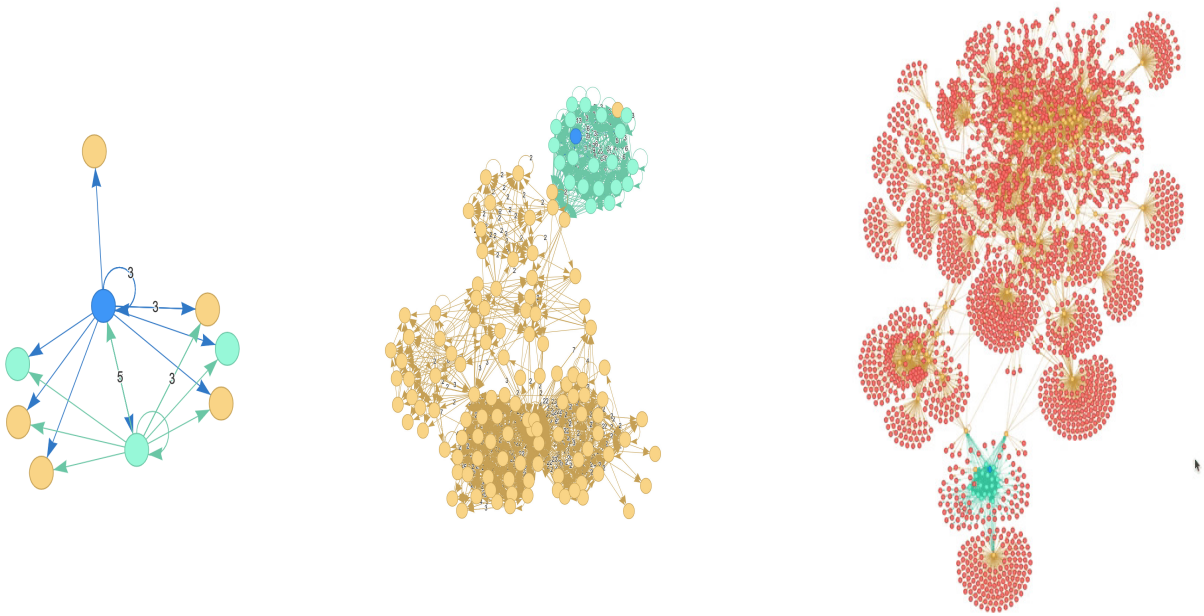


Figure – Crawling with multiple depths (from 1 to 3).

In this paper, the depth has been set to 1 due to the exponential growth in the number of links during crawling.

Graph Neural Networks

- A Graph Neural Network (GNN) is nothing more than a Neural Network taking as input a graph and extracting structural features from it
- It is possible to use typical Deep Learning layers after GNN layers, but not before because these layers don't know how to process graph data

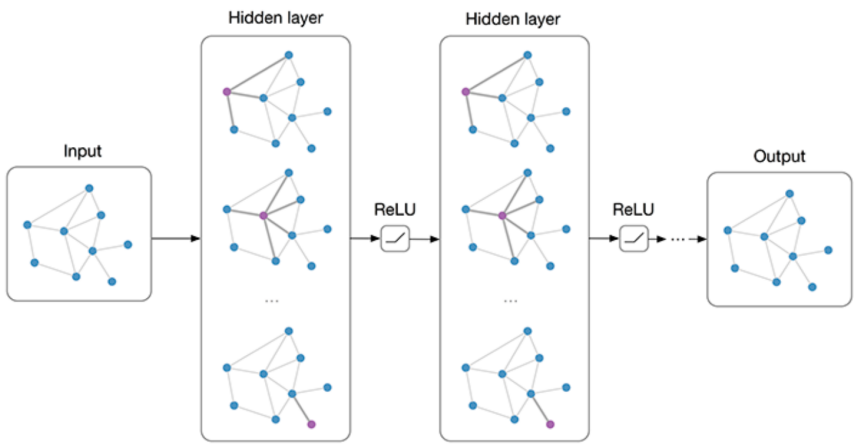


Figure – Graph Neural Network with 2 hidden layers.

Why do we need GNNs ?

- Graphs are unstructured data and traditional ML or Deep Learning methods are not efficient when used on such data
- In graphs, there is no notion of geographic position such as right, left, top, bottom

- Graphs contain important structural information that can be used for classification, regression or clustering
- Deep Learning techniques can leverage the graph structure to extract geometric features and improve predictions
- GNNs achieve state of the art performance in many use cases dealing with graphs (drug discovery, social networks, traffic, recommendations...)

Graph Convolutional Network

- Graph Representation Learning technique
- At each layer of the model, each node in the graph receives messages from its neighborhood node features + some learnable weights in order to learn the optimal coefficients to ponderate these features (fundamental concepts of Deep Learning)

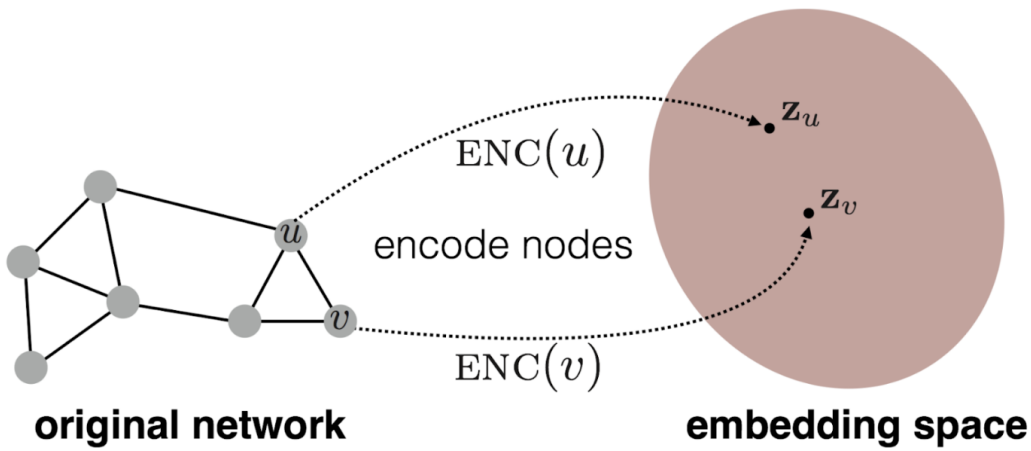
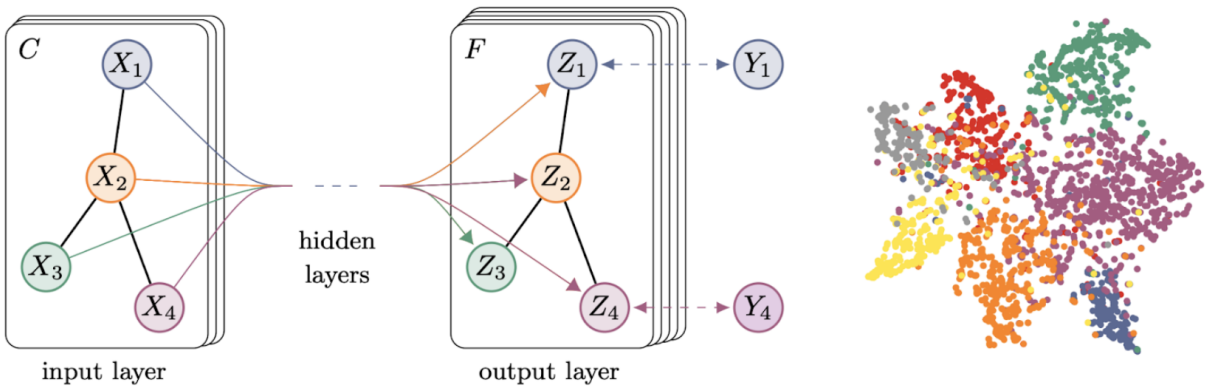


Figure – [Kipf et al. (2016)] Each convolutional layer will create node embeddings. Node embeddings will be near if they share common features.

Graph Convolutional Network

Layer-wise propagation rule of GCN



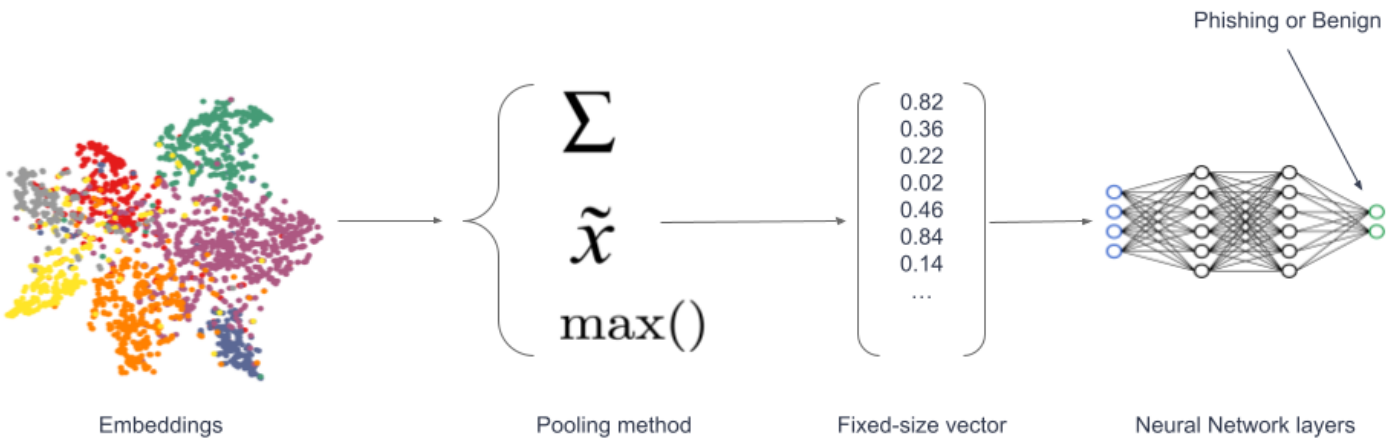
(a) Graph Convolutional Network

(b) Hidden layer activations

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

- $H^{(l)}$: Propagation at layer l, $H^{(0)} = X$ (the feature matrix)
- A : Adjacency matrix
- $\tilde{A}$: Adjacency matrix with self loops
- $W^{(l)}$: Weight matrix at layer l
- σ : Rectified Linear Unit non-linear activation function
- $\tilde{D}^{-\frac{1}{2}}$: Normalization coefficient applied to adjacency matrix

Graph Convolutional Network



- In the case of Graph classification, we want to apply a readout/pooling function on the node embeddings to obtain a final graph embedding.
- This graph embedding can be used in downstream Deep Learning models.
- In our example we apply MLP layers to classify the embeddings as phishing or benign.

phishGNN

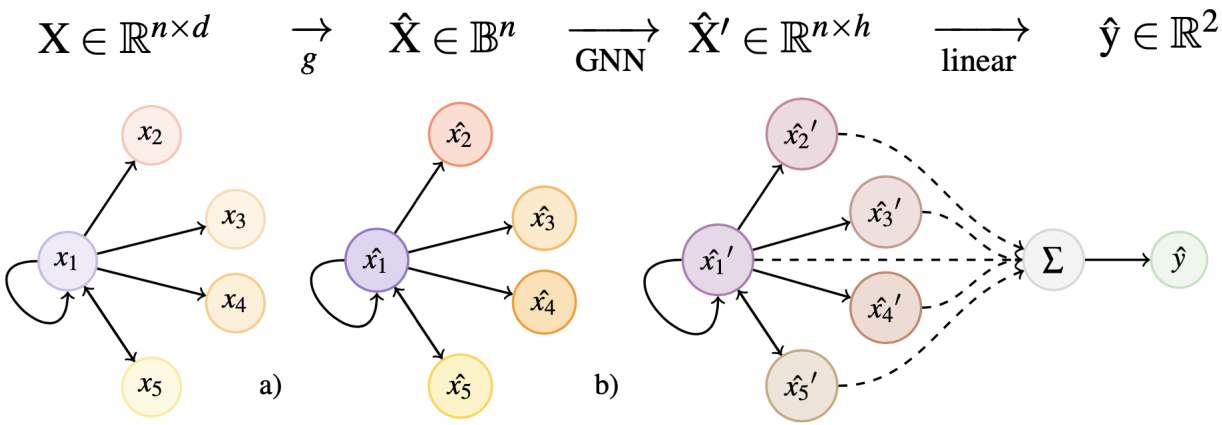


Figure 1: PhishGNN architecture comprises two steps: PRE-CLASSIFICATION (a) and MESSAGE-PASSING (b). Example using a graph with one root URL x_1 and 4 outgoing links $x_{2 \leq i \leq 5}$. The input feature matrix X is processed in these 2 steps to result in a prediction vector $\hat{y}$ containing the probability of the 2 classes.

Framework (2 steps)

1. Pre-classification : train a classifier with all known labels and then use it to predict every unknown label
2. Message-passing : gather all the predictions (0 or 1) from the previous step using a GNN, then apply pooling to reduce dimensionality and make a final prediction for the whole graph with a linear layer

Plan

Evaluation framework

Implemented models

- Graph Convolutional Network (GCN) => Thomas N. Kipf, Max Welling, Semi-Supervised Classification with Graph Convolutional Networks
- Graph Isomorphism Network (GIN) => Xu et al., How Powerful are Graph Neural Networks ?
- Graph Attention Network (GAT) => Petar Veličković et al., Graph Attention Networks
- GraphSAGE => William L. Hamilton et al., Inductive Representation Learning on Large Graphs
- ClusterGCN => Wei-Lin Chiang et al., Cluster-GCN : An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks
- MemPool => Matheus Cavalcante et al., MemPool : A Shared-L1 Memory Many-Core Cluster with a Low-Latency Interconnect
- Multi-Layer Perceptron (MLP) => Rosenblatt, The perceptron : a probabilistic model for information storage and organization in the brain.

Dataset

- Initially around 30k malicious URLs extracted from phishtank [a] and openphish [b]
- 2 filtering steps are then applied to these URLs :
  - filter the HTTP status code if it is an error (not in range 200-299) : reduce dataset by 85%
  - filter malicious URLs with Google Safe Browsing API to remove non-malicious websites : reduce dataset by 40%
- After filtering, the dataset contains 4633 high-quality URLs, where 2333 are phishing and 2300 are benign
- Benign URLs are extracted from the Alexa top 1 million sites dataset [c]

---

a. `https://phishtank.org/`
b. `https://www.openphish.com/`
c. `https://www.kaggle.com/datasets/cheedcheed/top1m`

Dataset

Model	Mean-Pool	Max-Pool	Add-Pool	Time
GIN	48±1.5	59±2.4	76±0.1	37.2
GAT	79±3.2	59±2.7	82±1.1	45.5
MemPooling	78±3.0	73±4.1	76±3.8	67.5
GCN_2	91±0.5	**93±0.2**	92±0.5	32.1
GCN_3	91±0.3	92±0.1	89±0.7	34.4
GraphSAGE	92±0.4	92±0.5	89±0.7	29.4
ClusterGCN	**93±0.3**	**93±0.6**	72±2.8	37.8

Figure 3: Model accuracy in % on test set for 10 epochs, for every implemented GNN. Each model is declined in three versions using multiple pooling methods (mean, max, add) as readout functions.

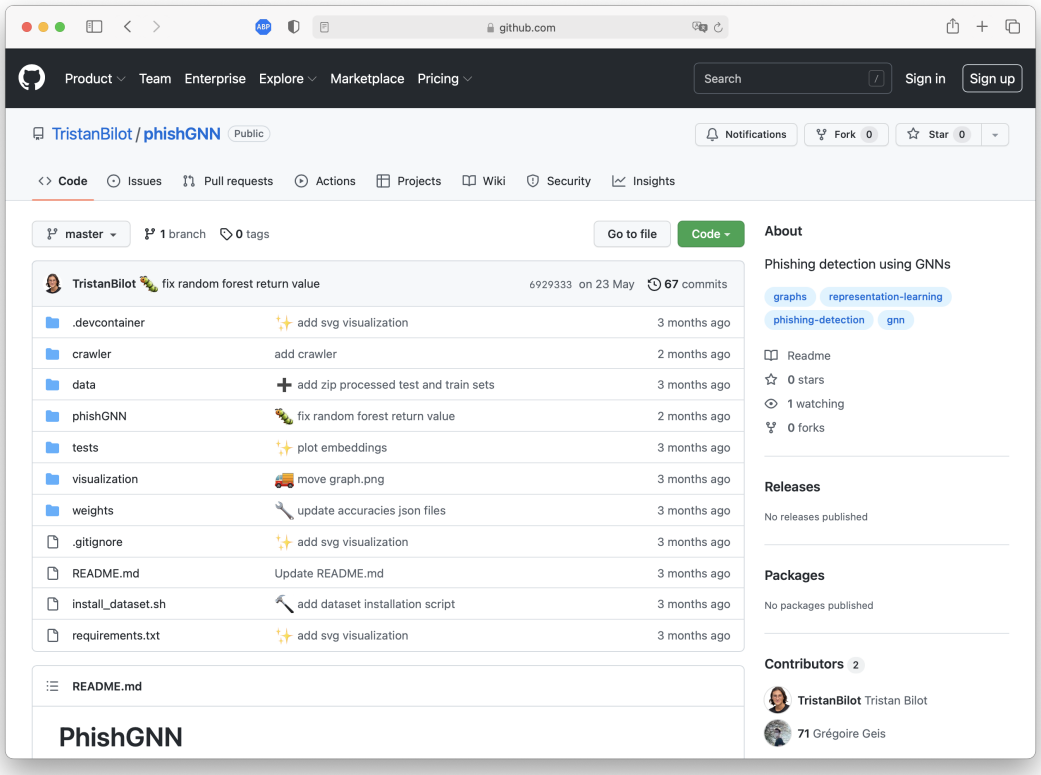
Model evaluation (without phishGNN)

Dataset

Dataset stats

- number of graphs : 4633
- mean of number of nodes/edges : 90/138
- max number of nodes/edges : 5185/5214
- min number of nodes/edges : 1/1

GitHub repository



The source code of the framework and experiments made in this study are available on GitHub [a].

a. https://github.com/TristanBilot/phishGNN

- ETSI TS 103097 standard is described by the meaning of a syntax derived from IETF RFC 2246 and from IEEE 1609.2-2012
- Syntax ambiguity ==> the implementation of these structures can lead to numerous interpretations
- Multiple implementations can be derived from this description (even those that do not respect the standard)

Ambiguity example 1

images/SignerInfo.png

SignerInfo

in the section dedicated to this structure, it is clearly explained that this type have to be only one of the three following :

1. self
2. certificate digest with SHA256
3. certificate digest with an other algorithm

Ambiguity example 2

images/SubjectAttribute.png

SubjectAttribute

- Does not support two SubjectAttributes of the same type
- A verification key and an assurance level are mandatory

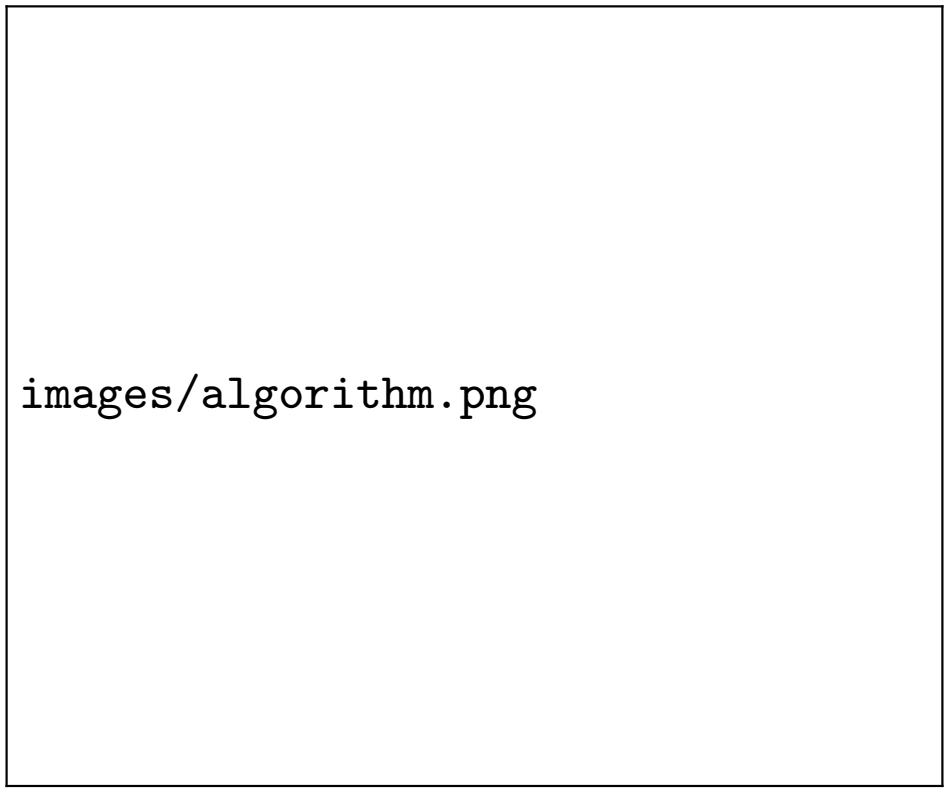
Overall context

- A C-ITS environment in which two ITS stations (ITSS) communicate
- Within a communication, a message is encoded by the first station, transmitted and finally decoded by the second one
- For authentication, the message contains the sender's certificate
- Our campaign study and compare the different encoding schemes : Binary in BigEndian form, PER, UPER, DER, BER, OER, COER, XER, CXER and EXER.

Experimental framework

- Two computers : sender and receiver
- Intel® Xeon® CPU E5-1607 v3 @ 3.10GHz (quad-core) with 8 Giga bytes of RAM
- ASN.1 implementation was realized using OSS Nokalva compiler (to Java code)
- Powerful computers, but fair comparison

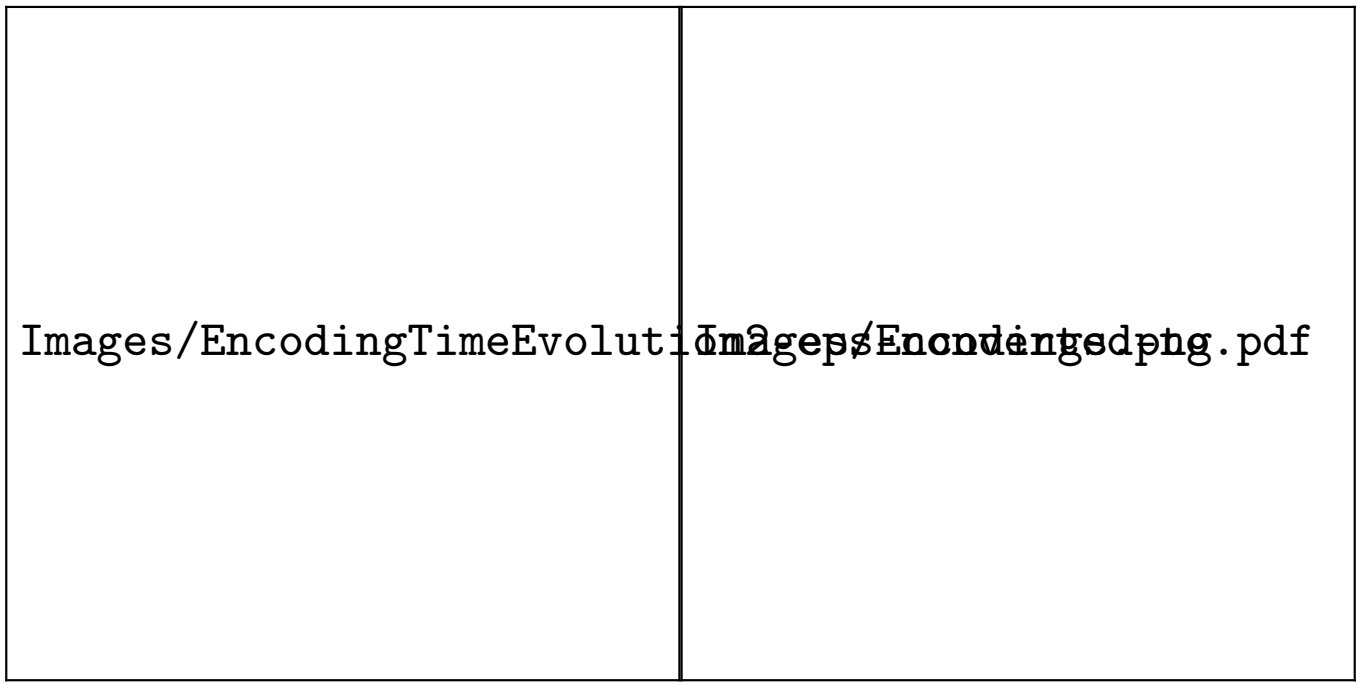
Experimental scenario



- $X = 10, 10^2, 10^3, 10^4, 10^5 and 10^6$

Plan

Encoding Times



(a) (b)

Figure – Encoding times

- (1) COER, (2) Binary, (3) OER, (4) UPER, (5) DER, (6) BER, (7) PER, (8) CXER, (9) XER, (10) EXER.

Decoding Times

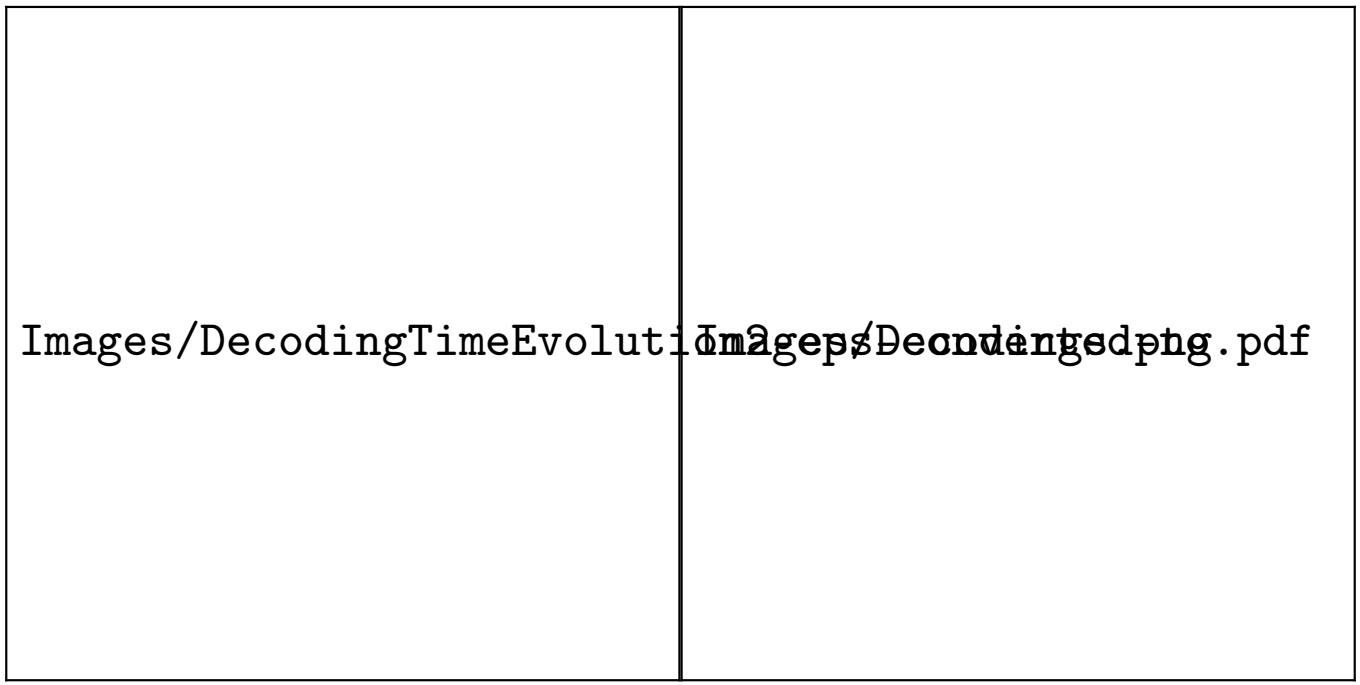


Figure – Decoding times

- (1) COER, (2) DER, (3) OER, (4) UPER, (5) BER, (6) PER, (7) Binary, (8) CXER, (9) XER, (10) EXER

certificate size

Encoding	Size average (bytes)	Size Var.	Size SD
UPER	187.3701	468.7203	21.64995
PER	192.3701	468.7203	21.64995
COER	198.3701	468.7203	21.64995
OER	198.3701	468.7203	21.64995
Binary	200.4964	479.9544	21.90786
DER	251.0994	632.6233	25.152
BER	251.0994	632.6233	25.152
CXER	1496.422	2496.244	49.96243
EXER	1880.13	5251.959	72.4704
XER	1924.137	5959.503	77.19782

Table – Statistics (Average, Variance and Standard Deviation) of the obtained results with 10^6 certificates

Ecoding and decoding speeds

Encoding $Speed = \frac{\delta length}{\delta time}$

1. XER : 54446.53 B/ms
2. CXER : 50432.18 B/ms
3. EXER : 39562.44 B/ms
4. COER : 14113.4 B/ms
5. DER : 12219.86 B/ms
6. BER : 12204.11 B/ms
7. Binary : 10938 B/ms
8. OER : 10810.92 B/ms
9. UPER : 9768.333 B/ms
10. PER : 8970.444 B/ms

Decoding

1. XER : 19370.19 B/ms
2. CXER : 17347.18 B/ms
3. COER : 12798.46 B/ms
4. DER : 11246.3 B/ms
5. EXER : 10332.09 B/ms
6. OER : 8686.24 B/ms
7. BER : 8267.766 B/ms
8. UPER : 7009.849 B/ms
9. PER : 5575.738 B/ms
10. Binary : 4517.666 B/ms

Plan

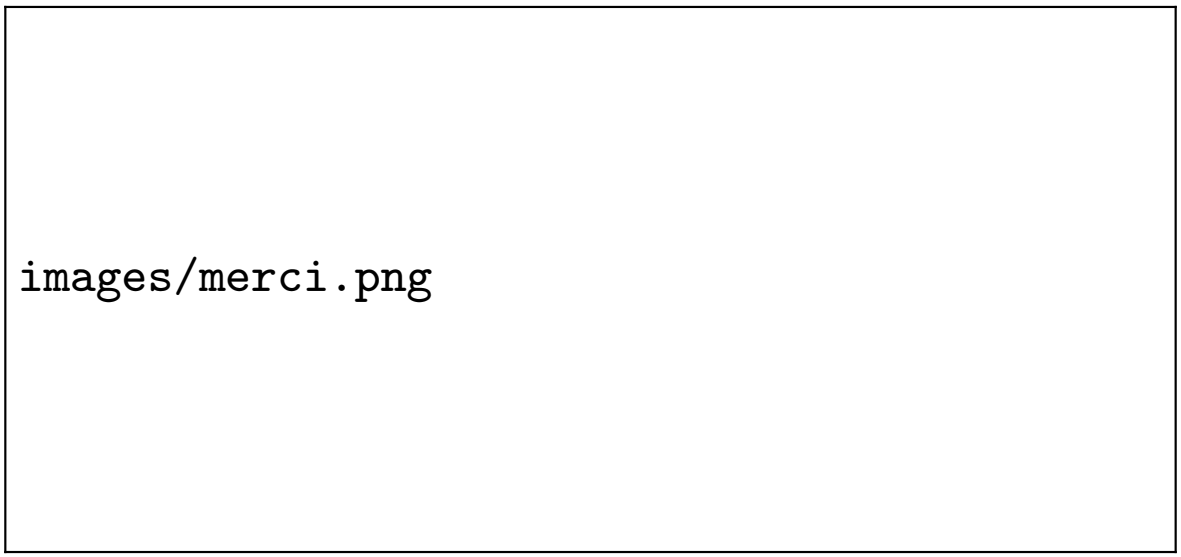
Conclusion and Future works

conclusion

- We provided an ASN.1 specification that ensure secure implementation of ETSI certificate
- We provided an extensive study on performance encoding schemes
- According to needs, the choice of the encoding scheme could be different :
  - COER realizes the best encoding and decoding times
  - UPER realizes enormous size savings
  - ASN.1 XML encodings are very fast comparing to other encodings but completely not adapted to this use case

Future works

- The submission of our ASN.1 definition to ETSI organism in order to be used for the standard description
- The specification of an ASN.1 definition for the ETSI secured message structure


images/merci.png