PhishGNN: A Phishing Website Detection Framework using Graph Neural Networks

Tristan Bilot, Grégoire Geis, Badis Hammi

PhishGNN: A Phishing Website Detection Framework using Graph Neural Networks

Tristan Bilot[1], Grégoire Geis[1] and Badis Hammi[1]
[1]*EPITA School of Engineering, France*
{*tristan.bilot, gregoire.geis, badis.hammi*}*@epita.fr*

Keywords: Phishing Detection, Graph Neural Networks, Deep Learning, Cybersecurity

Abstract: Because of the importance of the web in our daily lives, phishing attacks have been causing a significant damage to both individuals and organizations. Indeed, phishing attacks are today among the most widespread and serious threats to the web and its users. Currently, the main approaches deployed against such attacks are blacklists. However, the latter represent numerous drawbacks. In this paper, we introduce PhishGNN, a Deep Learning framework based on Graph Neural Networks, which leverages and uses the hyperlink graph structure of websites along with different other hand-designed features. The performance results obtained, demonstrate that PhishGNN outperforms state of the art results with a 99.7% prediction accuracy.

1 INTRODUCTION

In the era of the Internet, malicious URLs are a common threat to the Web users. Phishing aims at stealing sensitive information by fooling victims with falsified interfaces. In the case of phishing websites, attackers usually try to impersonate well-known and widely used services such as social media, banks and e-commerce websites. Such spoofed websites are often built from the same code base as the original site, which could make them difficult to detect at first glance. Thankfully, numerous other indicators can be used to differentiate benign and phishing websites. For instance, most phishing URLs tend to be very long, with multiple sub-domains and special characters. Domains are often hosted on suspicious hosts and use Secure Socket Layer (SSL) certificates delivered by non-trusted authorities. Since the beginning of these attacks, numerous systems have been implemented to try to overcome them. Some of these implementations use traditional techniques such as blacklists or URL lexical features' analysis. Nonetheless, blacklists suffer from multiple drawbacks like the need for human assistance to be updated and the lack of exhaustiveness. Furthermore, they cannot be used on unseen and hidden URLs. Other techniques leverage Machine Learning to train a model to classify websites based on a number of examples (Sahoo et al., 2017), (Benavides et al., 2020). However, in most approaches, the hyperlink structure of websites is not tackled.

In this paper we introduce PhishGNN, a framework that leverages and uses both hyperlink structural features along with other features that have been proven to be successful for phishing classification [1]. We also introduce features such as `is_same_domain` which are essential for differentiating two websites with the same structure. As many phishing websites redirect to legitimate ones, each link pointing to these websites has a different domain. However, on the legitimate website, these links are redirecting to the same domain, so the feature will be distinct in both cases and the model will learn how to differentiate them. We evaluated our approach through a real implementation. The performance results obtained demonstrate the efficiency and effectiveness of our approach in terms of detection accuracy and its capacity to outperform the existing detection approaches.

2 RELATED WORKS

The detection of phishing websites aims to classify whether websites are phishing or benign. Research in this area has increased sharply as the number of phishing websites has exploded in recent years. While advanced techniques have been proposed for this task, most solutions currently in production are based on blacklists (Sahoo et al., 2017). However, phishing

[1]https://archive.ics.uci.edu/ml/datasets/Phishing+Websites

websites become more and more complex and there is an urgent need for reliable and efficient techniques to detect them on demand, without human interaction.

2.1 Traditional Techniques

The most common technique used for the detection of phishing websites is the use of blacklists. However, this technique reveals numerous drawbacks, mainly: (1) it requires the manual curation of such a blacklist. (2) it requires the storage (space consumption) or the querying (time and computing resource consumption) of a blacklist. (3) crowdsourced blacklists like PhishTank are centralized and lack transparency. The resources consumption problem is addressed by the Google Safe Browsing API, notably used in Chromium and in Firefox, which provides both an online service and a small, downloadable database of truncated hashes [2].

Prakash et al. (2010) also show that it is possible to build blacklists that, based on their current entries, can predict new entries with no human involvement.

As lists are inherently flawed, other techniques have been proposed to detect phishing using human-defined heuristics, designed after identifying inherent characteristics of known phishing websites. Indeed, since obtaining legitimate domains requires compromising their corresponding entity, phishing websites often use patterns in the URL to appear like legitimate domains while being subtly different. Furthermore, Sonowal and Kuppusamy (2020) suggest that having symbols such as "–" and "@", or having more than three dots in the domain name is suspicious, and considers long URLs suspicious as well because they make it harder for users to read the significant part of the URL. Sonowal and Kuppusamy (2020) assign lower legitimacy scores to web pages that have low accessibility scores and to those whose lexical signature does not appear within the first results of search engines.

It was often assumed that legitimate websites are protected by Transport Layer Security protocol (TLS) and that phishing websites are not. However, this assumption no longer holds true because most recent phishing websites are also protected by TLS (82% in 2021 [3]). Despite this, new techniques exist to assess the legitimacy of TLS certificates (Sakurai et al., 2020).

Phishing websites can protect themselves against content-based phishing detection by obfuscating the content of their pages. In response, researchers have used image snapshots of suspicious pages to extract text content using optical character recognition (Dunlop et al., 2010), or to look up whether the suspicious page used logos that are typically associated with other domains (Wang et al., 2011).

Finally, it has been shown that using combinations of different techniques leads to more accurate results (Sonowal and Kuppusamy, 2020).

2.2 Machine Learning Techniques

Machine Learning (ML) and Deep Learning (DL) have known a real boom during the last decade and have been widely used in the phishing classification task (Sahoo et al., 2017), (Benavides et al., 2020). When using such techniques, the first step is usually to extract a set of features from the URL. Although Deep Learning models have the ability to learn features by themselves from raw data, it is challenging to exploit them directly because a website is not only defined by its raw HTML content. Many useful features can be extracted by hand from the website's URL in order to build a more powerful and robust model. Deep Learning can be used on top of the features to let the model learn other kinds of representations that could improve the classification accuracy. These features can be divided in three classes: lexical, content and domain features.

Lexical features: features that could be extracted from the URL as a string. Some examples are the URL length, the entropy, the number of special characters, the number of sub-domains, and so on.

Content features: features related to the HTML content of the web page. Such features are obtained by fetching the DOM of the pages and then processing it to extract useful information like the number of anchors, the presence of a form, the number of Javascript imports, and so on.

Domain features: features obtained from the domain name extracted from the URL. By requesting that domain, it is possible to extract features from the underlying server such as its location, the connection speed, "WHOIS" information, Secure Socket Layer (SSL) certificate and so on.

According to Sahoo et al. (2017), content-based and lexical-based features are mostly used in Machine Learning techniques, compared to host-based features, due to the complexity of extracting these ones.

Most state of the art approaches for phishing classification are URL-based. That is, they focus on the extraction of useful features directly from the raw

[2] https://developers.google.com/safe-browsing/v4/urls-hashing

[3] https://docs.apwg.org/reports/apwg_trends_report_q2_2021.pdf

URL. Some studies use traditional Machine Learning with hand-crafted features to make predictions (Jain and Gupta, 2018), while others prefer using Deep Learning to let the model learn the features on its own (Sahoo et al., 2017). Deep Learning methods have the benefit of avoiding human-assisted feature engineering and thus do not require the assistance of domain experts. Thanks to these benefits, numerous recent studies (Benavides et al., 2020) apply Deep Learning to URL classification. URL-based classification is a key process in the overall phishing classification task. This is due to the numerous lexical features possible to extract from a raw URL string.

Saxe and Berlin (2017) proposed eXpose, a solution based on a Convolutional Neural Network (CNN), where convolutions are applied to the raw URL at character-level. The convolutions are used to find patterns between characters that could lead to interesting hidden features. In the same context, URLNet (Le et al., 2018) is a framework where a character-level CNN is used in combination with a word-level CNN. It is stated that using word-level features along with character-level features achieve better results for the URL classification task. Other techniques such as HTMLPhish (Opara et al., 2020), take profit of CNNs to learn the semantic dependencies in the textual content of the raw HTML page. Using this architecture, they achieve 93% accuracy with no feature engineering required.

Similar to our solution, Tan et al. (2020) propose a graph-based detection system where hand-crafted features are extracted from the hyperlink structure of the webpage, achieving 97.8% accuracy using a C4.5 classifier. However, this implementation solely depends on human-created features that could be biased. The authors do not leverage the Deep Learning to let the model learn by itself the most useful features to differentiate benign and phishing examples. Furthermore, a dataset of only 1000 samples is used (500 benign, 500 phishing), resulting in a model that could difficultly generalize on new data.

To the best of our knowledge, the sole application of Graph Neural Networks to phishing detection is based on the HTML structure of the website (Ouyang and Zhang, 2021). In this approach, a graph is built from the HTML DOM and a GNN is fed with this graph. However, this method only relies on the HTML content, which could be easily stolen from benign websites in order to build perfect website copies. This method could thus be easily bypassed by cloning the HTML structure of legitimate websites.

Unlike previous approaches, our solution takes advantage of the internal links structure of the website, along with the traditional features that led to successful results as shown in previous papers. By analysing many phishing websites, we figured out that most of them use similar "href" patterns in `<a>`, `<form>` and `<iframe>` tags. These links are usually self-loops anchors (URLs starting by #) or outgoing links to external domains (usually pointing to a legitimate website like a bank or a social media). Such patterns are useful for phishing classification because a neural network can be trained to learn how to distinguish websites with different structures. Malicious websites could hardly bypass this detection system because most of the outgoing links present on these websites redirect to external websites from other domain names in order to fool victims by persuading them that the website is legitimate.

3 PROPOSED APPROACH

3.1 Graph Neural Networks

Graph Neural Networks (GNNs) represent a type of neural networks that takes graph data as input. Unlike other neural network architectures, GNNs can handle non-euclidean data with complex relations between objects. Most GNNs follow the message-passing framework (MPNN) (Gilmer et al., 2017) and can be considered as a generalization of convolutional neural networks (CNN) for graphs. This message-passing algorithm takes as input a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes $v_i \in \mathcal{V}$ and m edges $(v_i, v_j) \in \mathcal{E}$, where $\mathcal{G}$ could be directed or undirected. Each node and edge can store a vector of features, respectively named node and edge features. Generally, all this information is represented through three matrices:

- A: the graph-structure matrix, of shape $n \times n$ in the case of an adjacency matrix and $2 \times n$ for a CSR (Compressed Sparse Row) or COO (COOrdinates) sparse matrix
- X: the node features matrix of shape $n \times d$
- E: the edge features matrix of shape $m \times e$

where $n = |V|$, $m = |E|$, d and e are respectively the number of features per node and per edge.

The message passing framework consists of four steps, where steps 1 to 3 are implemented by one GNN layer and are repeated as many times as there are layers. Step 4 is a final step that should be applied after passing through every GNN layers.

1. MESSAGE: every node creates a message based on its node features and sends it to all neighbors.
2. AGGREGATE: nodes aggregate the incoming messages from every neighbor by using an aggregation function.

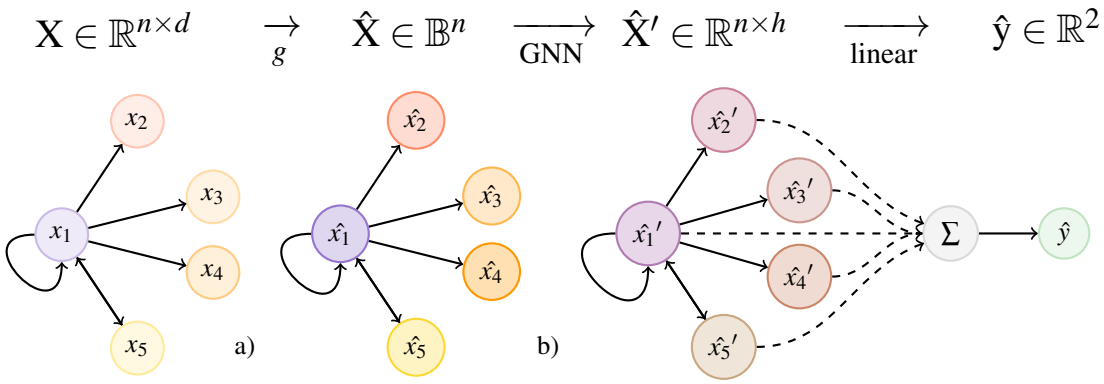

Figure 1: PhishGNN architecture comprises two steps: PRE-CLASSIFICATION (a) and MESSAGE-PASSING (b). Example using a graph with one root URL x_1 and 4 outgoing links $x_{2 \leq i \leq 5}$. The input feature matrix X is processed in these 2 steps to result in a prediction vector $\hat{y}$ containing the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

3. UPDATE: the old features of a node are updated by merging them with new features aggregated, creating node embeddings.
4. READOUT: combines every node embeddings into a representation that could be used in downstream Machine Learning algorithms for prediction.

Every step is generally a function with learnable parameters, that is, weight matrices and activation functions are used in the computation of both steps. One GNN layer usually corresponds to the propagation of features within a 1-hop neighborhood, so stacking n GNN layers will result in node features propagating up to n distant nodes. In each of these layers, every node gathers its neighbors' features to result in graph embeddings.

3.2 PhishGNN

We propose PhishGNN, a framework for websites classification (phishing or benign) based on Graph Neural Networks. This framework can be considered as an additional layer to GNN architectures. Therefore, it can be easily plugged-in existing GNN implementations. We use graphs to leverage the hyperlink structure of websites. In the context of GNNs, we consider the task of phishing websites classification as a node classification task, where the node to classify is a given URL and the other nodes represent every possible link coming from that URL until a user-defined depth. From these links, it is possible to build a graph where nodes represent URLs, and edges are the links between URLs, extracted either from `<a>`, `<form>` or `<iframe>` tags. More precisely, the graph is a rooted graph where the root node is the website to classify (named root URL). The input dataset (fed to our classifier) contains a list of root URLs, mapping to a label: phishing or benign. For each URL in the dataset provided, a feature vector is extracted, as well as a vector of all URLs (children URLs) going from that root URL. Features are also extracted for the children URLs. The feature vectors are used to build the feature matrix X. The children URLs are used to build the actual graph-structure matrix A.

In our approach, we suggest to train a model in a semi-supervised mode. The known labels are the actual root URLs and the unknown labels represent every child URL (i.e. we do not know if these URLs are phishing or not). Our contribution highly relies on the fact that knowing the label of every node around the root node makes that node much easier to classify. Given that labels are not known for every child URL, a classifier could be used to find an approximation for these labels. This classifier is trained on every supervised example available in the dataset and is then used for inference on all other unsupervised examples. Afterwards, using a traditional GNN with message passing will gather information from classified nodes to build the embeddings. We use pooling methods such as add, max or mean on top of the embeddings to reduce graph dimension to a single node embedding. A linear layer is used as a final layer to make a prediction.

As Figure 1 shows, the architecture is divided into two steps:

(a) PRE-CLASSIFICATION: initially, the graph comprises n nodes, where each node $x_i (1 \leq i \leq n)$ is a vector of d features extracted from the corresponding i^{th} URL. x_1 is the root URL node and every node $x_i (1 < i \leq n)$ represent a link coming from x_1. At this first step, a binary classifier is used to predict in a semi-supervised mode whether a node is phishing or benign, for each feature node $x_i (1 \leq i \leq n)$. The classifier is a function $g : \mathbb{R}^d \rightarrow \mathbb{B}$, where $\mathbb{B}$ is the Boolean domain. After

this step, the feature matrix X is transformed to a vector $\hat{X}$ containing respectively zeroes and ones for legitimate and phishing predictions.

(b) MESSAGE-PASSING: the predictions are then passed through a traditional message passing GNN with h hidden layers, to propagate the information in the graph and learn node embeddings. This results into a matrix $\hat{X}'$ where each node is an embedding vector of size h. A pooling method is used to reduce the dimension of node embeddings to a single node of shape $1 \times h$. Finally, a dot product is applied between this node and a linear layer of shape $2 \times h$, resulting into a vector $\hat{y}$ containing the probability of belonging into each class: phishing or benign.

The graph-structure matrix A is stored in memory using the COO format, which requires only $O(|\mathcal{E}|)$ memory space, i.e. it grows linearly according to the number of edges. The feature matrix X uses $O(|\mathcal{V}| \times d)$ memory as it stores fixed-size feature vectors for each node.

The propagation rule of PhishGNN with a Graph Convolutional Network (GCN) as MESSAGE-PASSING step is the same as the original GCN propagation rule:

$$f(H^{(l+1)},A) = f(H^{(l)},A) \tag{1}$$

$$f(H^{(l)},A) = \sigma(\hat{D}^{\frac{-1}{2}}\hat{A}\hat{D}^{\frac{-1}{2}}H^{(l)}W^{(l)}) \tag{2}$$

where A is the adjacency matrix, $H^{(l)}$ is the propagation at layer l, σ is the ReLU non-linear activation function (Rectified Linear Unit), $W^{(l)}$ is a weight matrix at layer l, $\hat{A}$ is the adjacency matrix with self loops such that $\hat{A} = A + I$ (I is the identity matrix), and $\hat{D}$ is the diagonal matrix of $\hat{A}$.
However, instead of starting with $H^{(0)} = X$ in the original GCN, PhishGNN applies the PRE-CLASSIFICATION step to X such that $H^{(0)} = g(X)$, where g is a Random Forest prediction function.

4 PERFORMANCE EVALUATION

4.1 Evaluation Framework

To train the model and later evaluate arbitrary inputs, raw features related to a given URL must be obtained. Unlike traditional classifiers operating on content features, PhishGNN must *crawl* web pages *recursively* to provide features for the pages referenced. Several existing crawlers were considered, but ultimately we implement a crawler specifically designed for PhishGNN with the following functionalities.

1. ROBUSTNESS: Servo's HTML and URL parsers were used, while domain names are parsed using Mozilla's Public Suffix List. `rustls` is used for establishing safe TLS connections. Pages that take more than 10 seconds to read, or that are over 1 MiB, or that lead to more than 10 redirects are dropped.
2. CONCURRENCY: multiple processes can operate on the same database at the same time, and each process contains workers which run in parallel (using OS threads) and concurrently (using asynchronous tasks).
3. DOMAIN-SPECIFICITY: two types of workers are available; *core* workers extract lexical and content features. *Domain* workers extract domain features. Thus, each domain is processed only once, no matter how many pages are hosted on it.
4. EXTERNAL STORAGE: the processing queue lives entirely on a database separate from the workers. This enables distributed workers to be stopped or resumed at will, and direct interaction with the database to, for instance, monitor progress.

Crawling websites can be a heavy and time-consuming task, which is why the crawler stops processing URLs after a specified *depth* is reached; in this study, we have set the crawling depth to 1 (that is, both pages of depth 0 and 1 are crawled for their features). A total of 25 features per URL are extracted during crawling. We classify the most important features as follows.

1. LEXICAL FEATURES: `is_ip_address`, `domain_length`, `domain_depth` (number of dots in the domain name), `dashes_count`, `has_at_symbol`.
2. CONTENT FEATURES: `is_valid_html`, `has_iframe`, `has_form_with_url`. References are added for `<a>`, `<form>`, and `<iframe>` elements with valid (i.e. statically known and leading to a valid HTTP or HTTPS URL after resolution) `href`, `action`, and `src` attributes.
3. DOMAIN FEATURES: `is_cert_valid` (i.e. active and accepted by `rustls`), `cert_reliability` (computed using the duration of the certificate and whether its issuer is trusted), `has_whois`, `domain_age` (seconds between the last update date and the expiry date).

After extraction, features are exported to a file which can be read and pre-processed in Python. To better understand the underlying structure of each website, we have developed a tool to visualize every graph from the dataset. In Figure 2, two crawled web pages, with different structures, are represented as graphs.

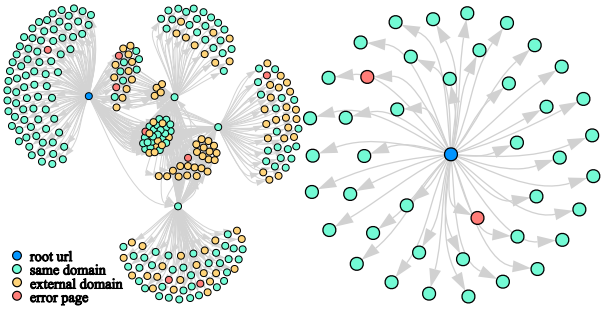

Figure 2: Graph representation of two websites after crawling with depth=1. Graph on the left contains multiple children URLs already crawled in previous iterations so their children are inserted in the graph as nodes of depth 2. Graph on the right contains children URLs never crawled before. Node in dark blue is the root URL, nodes in cyan and yellow are respectively URLs from the same domain and different domain, while red nodes are URLs returning an error code (HTTP status not in range 200-299).

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.

4.2 Dataset

Finding a reliable public phishing dataset is fairly challenging because the lifetime of phishing websites is really short (few days or weeks). Hence, we have built a dataset based on around 30,000 malicious URLs, extracted from public phishing blacklists such as OpenPhish[4] or PhishTank[5]. However, most of these URLs redirect to 404 error pages as the corresponding websites are now out of service. The first filtering operation to apply on the dataset is thus to check that every website is responding with a successful HTTP code (i.e. in the range 200-299). This step has reduced the dataset size by 85%. Furthermore, some of the filtered URLs are labeled incorrectly. Indeed, totally legitimate websites like wikipedia.org or baidu.com are sometimes classified as phishing instead of benign. These incorrect classifications could lead to a biased model and therefore to incorrect predictions. To prevent this, we used the Google Safe Browsing API in order to filter the dataset. Using this service on every URL from the dataset improves the reliability of each training example and brings a fairly better data quality but also removes a significant amount of data. This filtering step reduces the size of the dataset again by around 40% but has proven to be profitable. Furthermore, only websites containing at least a `<form>`, `<input>` or `<textarea>` tag are used for training. Indeed, we assume that phishing web pages usually request the user's personal information. A web page not containing such HTML tags is therefore not trying to steal any information.
Benign URLs are extracted from the Alexa top 1 million sites dataset [6]. The same filtering process is applied, except for the Safe Browsing API filter. We use approximately the same number of training examples in both classes in order to obtain a balanced dataset.

After the filtering steps, the overall dataset contains 4633 high quality URLs: 2300 benign and 2333 phishing. Graph matrices are built from the crawled URLs of the dataset. These graphs possess the following statistics: 90 average and 31 median nodes, ranging from 1 to 5185 nodes, 138 average and 45 median edges, ranging from 0 to 5214 edges.

Model	Mean-Pool	Max-Pool	Add-Pool	Time
$\text{PhishGNN}_{\text{-GIN}}$	52.4	53.2	71.2	23
$\text{PhishGNN}_{\text{-GAT}}$	88.9	62.1	95.0	90
$\text{PhishGNN}_{\text{-MemPooling}}$	75.8	99.2	98.0	23
$\text{PhishGNN}_{\text{-GCN}_2}$	**99.7**	**99.7**	99.1	**20**
$\text{PhishGNN}_{\text{-GCN}_3}$	99.7	99.7	99.2	22
$\text{PhishGNN}_{\text{-GraphSAGE}}$	99.6	99.6	99.6	17
$\text{PhishGNN}_{\text{-ClusterGCN}}$	99.7	99.7	97.2	24

Figure 4: Accuracy of PhishGNN framework on test set for 1 epoch using a Random Forest setting. PhishGNN_{GCN_2}, PhishGNN_{GCN_3} and $\text{PhishGNN}_{\text{ClusterGCN}}$ achieve best results with 99.7% accuracy.

4.3 Numerical Results and Discussion

4.3.1 Evaluation of Existing GNNs

A total of 7 well-known GNNs have been implemented and trained on the crawled dataset. Every model was implemented[7] in Python using the PyTorch Geometric library. In this section, we describe the benchmarking performances of the models based on the raw features, without considering the PhishGNN implementation. Models performance is measured using 10-fold cross-validation during training. Cross-validation allows to test the model on every dataset example and thus gives a better indication of how well the model performs on unseen data. For

[4] https://www.openphish.com/
[5] https://phishtank.org/
[6] https://www.kaggle.com/datasets/cheedcheed/top1m
[7] Experiments done in this paper are available on GitHub: https://github.com/TristanBilot/phishGNN.

each GNN architecture, the network is trained for 10 epochs using Adam optimizer with a batch size of 32. Hyperparameters have been tuned using a validation set during training. The training starts with a learning rate of 10^{-2} and is decreased by 5% every 3 epochs, most of the time resulting in a better performance when set to 9025.10^{-3}. The loss is computed at each epoch using cross-entropy. Implemented models are GIN (Xu et al., 2018), GAT (Veličković et al., 2017), MemPooling (Ahmadi, 2020), GCN (Kipf and Welling, 2016), GraphSage (Hamilton et al., 2017) and ClusterGCN (Chiang et al., 2019). GCN_2 and GCN_3 are respectively implementations of GCN with 2 and 3 GCN layers. Training has been done on a NVIDIA Tesla K80 GPU using 16, 32 and 64 hidden neurons, where the setting with 32 hidden neurons gave the best accuracy on the validation set. The obtained results are therefore based on models trained with hidden layers of size 32. Corresponding accuracies (mean $\pm$ standard deviation) and the average execution times are listed in Figure 3.

4.3.2 Evaluation of PhishGNN

In this section, we are interested in benchmarking PhishGNN framework with every GNN architecture implemented previously. Traditional Machine Learning techniques are also evaluated in order to find the best classifier to integrate with PhishGNN. As Figure 5 describes, most traditional Machine Learning methods achieve equivalent or even better results than the previous GNNs. Thereby, the Random Forest (i.e. the classifier with best accuracy) is chosen as the default classifier used in the PhishGNN architecture. By combining Random Forest predictions as PRE-CLASSIFICATION step and GCN_2 as MESSAGE-PASSING step, we outperform every other result by a large gap with an accuracy of 99.7%. The accuracy has been computed according to Equation 3:

$$Acc = \frac{C}{N} \tag{3}$$

where C is the number of correct predictions and N is the total number of predictions. A detailed analysis of true and false positives/negatives is demonstrated in Figure 6.

	Benign	Phishing	Total
Benign	688	3	691
Phishing	2	802	804
Total	690	805	1495

Figure 6: Confusion matrix for a test set of 1495 examples (30% of the overall dataset).

As Figure 4 shows, we achieve high scores with every pooling method in only one epoch. As

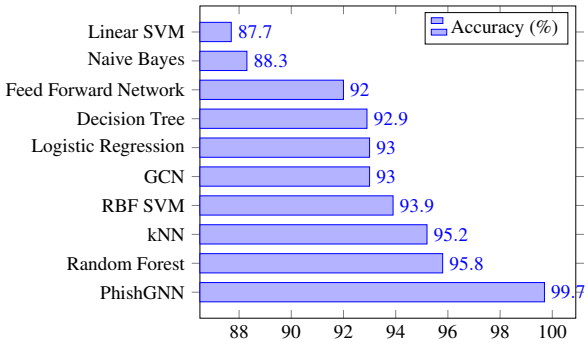

Figure 5: Classification accuracies between traditional Machine Learning methods, GCN and PhishGNN.

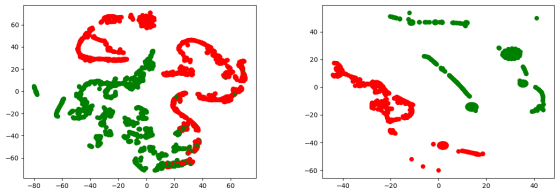

Figure 7: Embeddings of two models trained on our dataset. GCN_2 without PhishGNN (left) and with PhishGNN (right). Green: Benign; Red: Phishing

predictions are already pre-computed in the PRE-CLASSIFICATION step, there is no need to train the GNN multiple times, as we want to propagate the information one time to obtain node embeddings.

To better understand the model predictions, node embeddings have been extracted directly after the pooling step and are plotted in Figure 7, using the T-distributed Stochastic Neighbor Embedding (TSNE) dimension reduction technique. Although the traditional GCN achieves great classification results, we see in embedding space that the model fails to delimit many nodes. However, thanks to the the PRE-CLASSIFICATION step in PhishGNN, node embeddings are more grouped and classes can be delimited by a straight line, which leads to a better classifications.

5 CONCLUSION AND FUTURE WORKS

To the best of our knowledge, we introduced the first Graph Neural Network framework applied to website hyperlink structure for the phishing classification task. Our experiments has shown that GNNs directly applied on the website graph structure is less efficient than traditional Machine Learning methods applied to features. However, by leveraging the semi-supervised

structure of the graph, a classifier can be trained on supervised examples and make predictions on unsupervised ones. The semi-supervised predictions are then taken by a GNN as new input features and after message-passing, outperforms both traditional and Machine Learning techniques. The best accuracy has been achieved using a GCN combined with a Random Forest classifier. Furthermore, our approach is easily pluggable with any GNN architectures or other downstream classification methods. Hence, can be adjusted and improved in future works.

For future works we will focus on the establishment of a larger dataset, that contains more diverse examples. This dataset will be used in further research to improve benchmarking capabilities for phishing classification based on GNNs. We will also focus on improving the accuracy of our approach via leveraging edge features in the graph.

REFERENCES

Ahmadi, A. H. K. (2020). *Memory-based graph networks*. PhD thesis, University of Toronto (Canada).

Benavides, E., Fuertes, W., Sanchez, S., and Sanchez, M. (2020). Classification of phishing attack solutions by employing deep learning techniques: A systematic literature review. *Developments and advances in defense and security*, pages 51–64.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. (2019). Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266.

Dunlop, M., Groat, S., and Shelly, D. (2010). Goldphish: Using images for content-based phishing analysis. In *2010 Fifth international conference on internet monitoring and protection*, pages 123–128. IEEE.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Jain, A. K. and Gupta, B. (2018). Phish-safe: Url features-based phishing detection system using machine learning. In *Cyber Security*, pages 467–474. Springer.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Le, H., Pham, Q., Sahoo, D., and Hoi, S. C. (2018). Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*.

Opara, C., Wei, B., and Chen, Y. (2020). Htmlphish: enabling phishing web page detection by applying deep learning techniques on html analysis. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Ouyang, L. and Zhang, Y. (2021). Phishing web page detection with html-level graph neural network. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 952–958. IEEE.

Prakash, P., Kumar, M., Kompella, R. R., and Gupta, M. (2010). Phishnet: predictive blacklisting to detect phishing attacks. In *2010 Proceedings IEEE INFOCOM*, pages 1–5. IEEE.

Sahoo, D., Liu, C., and Hoi, S. C. (2017). Malicious url detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*.

Sakurai, Y., Watanabe, T., Okuda, T., Akiyama, M., and Mori, T. (2020). Discovering httpsified phishing websites using the tls certificates footprints. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 522–531. IEEE.

Saxe, J. and Berlin, K. (2017). expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. *arXiv preprint arXiv:1702.08568*.

Sonowal, G. and Kuppusamy, K. (2020). Phidma–a phishing detection model with multi-filter approach. *Journal of King Saud University-Computer and Information Sciences*, 32(1):99–112.

Tan, C. L., Chiew, K. L., Yong, K. S., Abdullah, J., Sebastian, Y., et al. (2020). A graph-theoretic approach for the detection of phishing webpages. *Computers & Security*, 95:101793.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, G., Liu, H., Becerra, S., Wang, K., Belongie, S. J., Shacham, H., and Savage, S. (2011). *Verilogo: Proactive phishing detection via logo recognition*.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.