An end-to-end approach for the detection of phishing attacks

Badis Hammi, Tristan Billot, Danyil Bazain, Nicolas Binand, Maxime Jaen, Chems Mitta, Nour El Madhoun

An End-to-End Approach for the Detection of Phishing Attacks

Badis Hammi[1], Tristan Billot[2], Danyil Bazain[3], Nicolas Binand[3], Maxime Jaen[3], Chems Mitta[3], and Nour El Madhoun[45]

[1] SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France
badis.hammi@telecom-sudparis.eu
[2] Université Paris-Saclay, France
tristan.bilot@universite-paris-saclay.fr
[3] EPITA Engineering School, France
danyil.bazain, nicolas.binand, maxime.jaen, chems.mitta@epita.fr
[4] LISITE Laboratory, ISEP, 10 Rue de Vanves, Issy-les-Moulineaux, 92130, France
nour.el-madhoun@isep.fr
[5] Sorbonne Université, CNRS, LIP6, 4 place Jussieu 75005 Paris, France

Abstract. The main approaches/implementations used to counteract phishing attacks involve the use of crowd-sourced blacklists. However, blacklists come with several drawbacks. In this paper, we present a comprehensive approach for the detection of phishing attacks. Our approach uses our own detection engine which relies on Graph Neural Networks to leverage the hyperlink structure of the websites to analyze. Additionally, we offer a turnkey implementation to the end-users in the form of a Mozilla Firefox plugin.

1 Introduction

Phishing is one of the most common forms of cyber crime on the web, especially in the last years as the Figure 1 shows. According to Verizon's 2023 Data Breach Investigations Report [1] 36% of all data breaches involved phishing. Also, according to Forbes[6] over 500 million phishing attacks have been reported in 2022. This number has been more than doubled compared to 2021, which is not surprising, considering that it's one of the easiest scams to execute. According to the Cybersecurity and Infrastructure Security Agency (CISA)[7], phishing is a form of social engineering in which a cyber attacker poses as a trustworthy colleague, acquaintance, or organization to lure a victim into providing sensitive information or network access. The lures can come in the form of a crafted email, text message, or even a phone call. However, the email factor remains the most used one. It is estimated that 3.4 billion malicious emails are sent everyday[8]. According to [2] the direct financial loss from successful phishing attacks increased by

[6] www.forbes.com/advisor/business/phishing-statistics/
[7] cisa.gov/sites/default/files/2023-02/phishing-infographic-508c.pdf
[8] www.itgovernance.co.uk/blog/51-must-know-phishing-statistics-for-2023

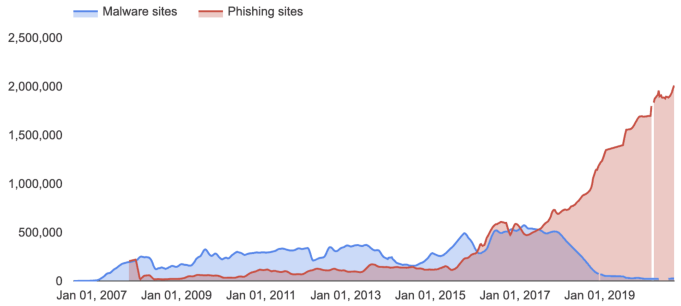


Fig. 1: Number of phishing websites observed between 2007 and 2021 according to Google Safe Browsing[9]

76% in 2022. Hence, the detection of phishing attacks remains among the most important/sensitive tasks to ensure the security of web users.

Unfortunately, despite the existing academic works that aim for the detection of phishing attacks, most of the deployed/implemented solutions rely on collaborative blacklists [3] [4]. We present in this paper the continuation of our previous work. In [5] we discussed the use of Graph Neural Networks for the detection of phishing attacks. More precisely, we showed that GNNs directly applied to the website graph structure are less effective compared to traditional machine learning methods applied to features. In this work, we demonstrate that through using the semi-supervised structure of the graph built using a website's structure, a classifier can be trained on supervised data and provide predictions on unsupervised ones.

We highlight the contributions of this paper as follows: (1) We propose an end-to-end approach for the detection of phishing attacks. Unlike the existing works, our detection engine leverages the hyperlink structure thanks to Graph Deep Learning, along with many other hand-crafted features learned with traditional Machine Learning. (2) We provide a ready to use solution, available for the end-users through a Mozilla Firefox plugin. (3) We provide the source codes of our implementation (the plugin for the client and the detection engine for the server).

To the best of our knowledge, our work is the first to introduce a comprehensive end-to-end phishing detection solution built upon graph neural networks.

2 Related works

Despite the proposal of numerous phishing detection techniques in academia, most of currently operational solutions rely on crowd-sourced blacklists [3]. In this section we discuss the main related works in academia and commercial solutions.

[9] https://transparencyreport.google.com/safe-browsing/

2.1 Academic related works on phishing detection

Traditional Techniques The predominant method employed to identify phishing websites involves the use of blacklists. Nevertheless, this approach is associated with several limitations, namely: (1) it needs the creation and maintenance of such blacklists, making it susceptible to zero-day attacks and dependent on human intervention. (2) it demands either storage capacity (resulting in space consumption) or frequent querying (leading to time and computing resource consumption) of a blacklist. (3) crowd-sourced blacklists, such as PhishTank are centralized and lack transparency. The resource consumption problem was tackled by the Google Safe Browsing API[10]. This API is prominently employed in Chromium and serves as a fallback in Firefox. It enables clients to manage a compact local database comprising only truncated hashes of malicious Uniform Resource Locators (URLs). However, this solution remains vulnerable to zero day attacks (new phishing domain names).

Because of the limits of blacklist based approaches, different other techniques have been proposed, based on human-defined heuristics, and designed after identifying inherent characteristics of known phishing websites. Indeed, phishing websites often use patterns in the URL to make them look like legitimate domains, while being subtly different. This can be done by confusing users with slightly different names (e.g. targeting "foobar.com" using the domain name "foo-bar.com"), by using subdomains of trusted entities (e.g. "foobar.example.com") or by including keywords related to the trusted entity in the path section of the URL (e.g. "example.com/foobar" [6]). Other lexical features derived from the URL can be useful. *Sonowal et al.* [7] 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.

Machine Learning Techniques 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 URL. Some works [8] employ conventional machine learning techniques, incorporating manually designed features for prediction. In contrast, others [3] opt for deep learning methods, allowing the model to autonomously learn features. The use of deep learning offers the advantage of avoiding human-assisted feature engineering, eliminating the need for domain expert intervention. Consequently, many recent studies [9] leverage deep learning for URL classification, considering it a crucial step in the broader task of phishing classification. This importance arises from the multitude of lexical features that can be extracted from a raw URL string. *Saxe et al.* [10] introduced eXpose, a solution based on a Convolutional Neural Network (CNN). *Le et al.* [11] proposed URLNet, a framework that integrates a character-level CNN with a word-level CNN.

[10] https://developers.google.com/safe-browsing/v4

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 [12]. 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 prefect website copies. This method could thus be easily bypassed by cloning the HTML structure of legitimate websites.

In contrast to prior works, our approach capitalizes on the internal links structure of the website, in conjunction with the conventional features that have demonstrated success in previous approaches. By analyzing multiple phishing websites, we observed that most of them employ 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 prove valuable for phishing classification, since a neural network can be trained to discern distinct structures among websites. 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.

2.2 Commercial phishing detection solutions

Most of the commercial solutions that aim to protect users from phishing attacks are available as web browser plugins. In this section, we present the most used ones. Table 1 presents a comparison of these solutions with our work.

Google Safe Browsing is mainly available on Google Chrome web browser and relies on an updated crowd-sourced blacklist of domain names. However, to protect users' privacy Chrome sends only a fraction of the URL to be checked to Google's server, not the full URL [4]. McAfee WebAdvisor is another browser extension designed to help users browse safely by alerting them to potentially malicious or dangerous websites. Similar to the Chrome plugin, McAfee WebAdvisor relies on crowd-sourced blacklists. However, it differs from Chrome in its ability to evaluate search engine results and provide indications of website security directly in these results [13]. Norton Safe Web is also a web browser plugin that relies on a crowd-sourced blacklist for phishing detection. Furthermore, it integrates Norton's threat intelligence network, which enables it to identify other online threats, such as malware and trackers [14]. The Avast Online Security plugin, like the previously described plugins use crowd-sourced blacklists to detect phishing websites. However, it relies on a cloud-based architecture, which is continually updated, guaranteeing real-time detection of emerging threats [15]. There exist some other browser extensions like Bitdefender TrafficLight or Kaspersky Protection. However, they all rely on crowd-sourced blacklists like the previously described solutions. Indeed, to the best of our knowledge, most of the commercial solutions rely on collaborative blacklists making these solutions dependent on these lists which are often flawed [5], consequently im-

Table 1: Comparison of commercial solutions for phishing detection (✓: Yes, ✗: No)

Approach	Crowd-sourced blacklist-based	Real-time detection capability	Vulnerable to zero day attacks	Transparency regarding blacklisting criteria	Dependency on continuous human intervention	Sensitivity to human bias
Google Safe Browsing	✓	✗	✓	✗	✓	✓
Avast Online Security	✓	✓	✓	✗	-	-
McAfee WebAdvisor	✓	✗	✓	✗	✓	✓
Norton Safe Web	✓	✗	✓	✗	✓	✓
Bitdefender TrafficLight	✓	✗	✓	✗	✓	✓
Kaspersky Protection	✓	✗	✓	✗	✓	✓
Our solution	✗	✓	✗	✓	✗	✗

pacting the users' experience. In our solution we rely on our own detection engine PhishGNN.

3 Proposed approach and implementation

The architecture we propose is similar to the Online Certificate Status Protocol (OCSP). As the Figure 2 shows, the architecture includes two additional entities compared to the conventional client-server architecture; a web plugin and a detection server (PhishGNN responder). No modifications to the web client or to the web server are required. More precisely, the web plugin acts as a proxy and intercepts the HTTP request that the web browser creates. The proxy extracts the domain name from the request and sends it to the detection responder. Next, the detection engine (within the detection responder) analyses the domain name and sends a boolean response to the proxy. If the detection responder's response indicates that the domain name is a phishing domain name, then, the plugin blocks the request. However, if the detection responder's response indicates that the domain name is safe, the proxy forwards the original HTTP request to the web server. In the following we describe the different parts of the architecture and the implementation choices we made[11].

3.1 Detection engine

Our detection engine relies on an extension of PhishGNN, an approach that we proposed for the classification of websites as phishing or benign [5]. Our detection engine leverages a Graph Neural Network (GNN) model to capture complex patterns hidden in the underlying hyperlink structure of web pages[12].

[11] A video that shows the implementation of our approach (with a plugin developed on Mozilla firefox) is available on: `https://youtu.be/SNik7Du3Mk8`

[12] The source code of our detection engine is available on: `https://github.com/TristanBilot/phishGNN`.

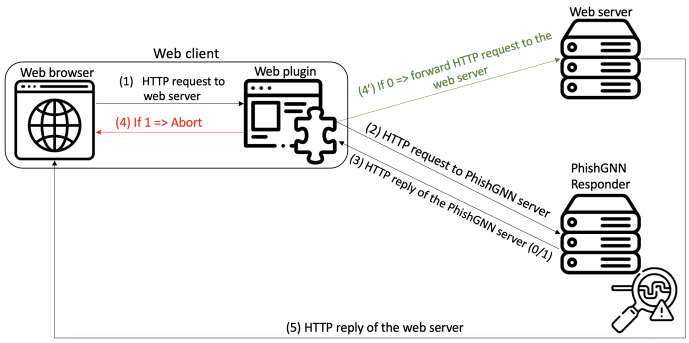
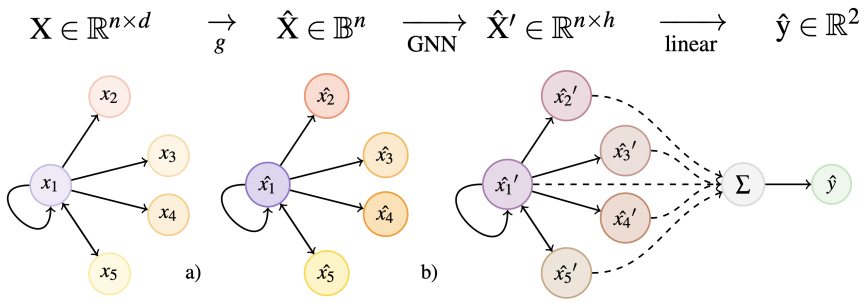


Fig. 2: System architecture of the end-to-end approach for the detection of phishing attacks

More precisely, we consider the task of phishing websites classification as a node classification task. In this context, the node to classify is a specific URL, and the other nodes represent each potential link originating from that URL. 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. Hence, the graph is structured as a rooted graph, with the root node identified as the website to be classified, commonly referred to as the root URL. For each root URL, a feature vector is derived, along with a vector encompassing all URLs going from the root URL (referred to as children URLs). Features are similarly extracted for these children URLs. Subsequently, the features' vectors contribute to the construction of the features matrix X. The children URLs are used to build the actual graph-structure matrix A.

In our approach, we propose training the model in a semi-supervised mode. The known labels pertain to the actual root URLs, while the unknown labels encompass every child URL—meaning it is unknown whether these URLs are phishing or not. Our approach heavily relies on the premise that having labels for every node around the root node significantly facilitates the classification of that root node. Since labels are unavailable for every child URL, we employ a random forest classifier to infer these labels. This classifier is trained on supervised examples in the dataset and is subsequently used for inference on all other examples. Following this, a GNN with message passing gathers information from classified nodes to construct embeddings. Pooling methods such as add, max, or mean are applied to these embeddings to reduce the graph dimension to a single node embedding. Finally, a linear layer is employed as the last layer for graph classification. The Figure 3 describes how the detection engine follows two steps:

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-



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.

Fig. 3: PhishGNN architecture

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 $g : \mathbb{R}^d \to \mathbb{B}$, that maps node features of size d to a prediction in the Boolean domain $\mathbb{B}$. After this step, the feature matrix X is transformed to a vector $\hat{\mathrm{X}}$ containing respectively zeroes and ones for legitimate and phishing predictions.

Message-passing The predictions are subsequently fed into a conventional message-passing GNN with h hidden layers. This process allows for the propagation of information throughout the graph and the learning of node embeddings. The outcome is represented as a matrix $\hat{\mathrm{X}}'$ where each node is an embedding vector of size h. A pooling method is used to reduce the dimension of graph embedding to a single node of shape $1 \times h$. Subsequently, a dot product is conducted between this node and a linear layer with a shape of $2 \times h$, yielding to a vector $\hat{\mathrm{y}}$ that encapsulates the probability of belonging to each class: phishing or benign.

Once implemented, our detection engine must crawl web pages recursively to extract features for the referenced webpages. Despite the existence of multiple web crawlers, we chose to implement our own crawler in order to meet the requirements of PhishGNN. The source code of the crawler is made available in [5]. Figure 4 exhibits some of the extracted features for every URL. We classify the features that the crawler extracts as (1) lexical features, (2) content features, and (3) domain features. Table 2 describes some of these features. Once features have been extracted by the crawler, they are exported to a `CSV` file which can then be read and pre-processed in Python. The detection server caches every domain name that it analyzes. In order to avoid the issue of aging domain names to use them for phishing attacks, a configurable Time To Live (TTL) value is assigned to the cached domain names.

In summary, from the detection responder's perspective, upon receiving an incoming request, it initially extracts the domain name from the request. A hashtable lookup is then performed to determine whether this domain is already present in the cache memory. If the domain is found in the cache, the server sends

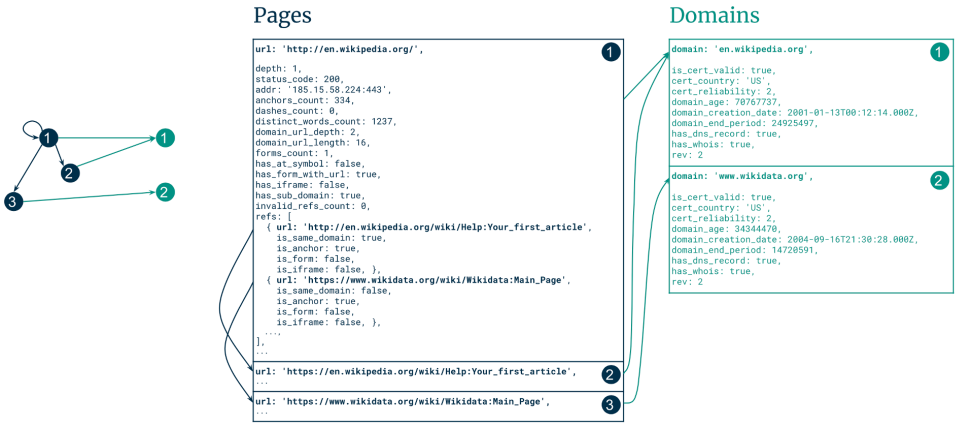


Fig. 4: Example of some extracted feature for every URL

Table 2: Classification of the features

Lexical features	Content features	Domain features
`is_https` (is the URL scheme "https"), `is_ip_address` (is the domain an IP address in any form), `domain_length` (length of the domain name, including subdomains and Top Level Domain (TLD)), `domain_depth` (number of dots in the domain name), `has_subdomain` (`domain_depth` $\geq$ 2), `dashes_count` (number of dash characters in the domain name), `has_at_symbol` (contains "@"), `is_same_domain` (false if the URL domain is not the same as the root URL)	`is_valid_html` (false if the response body contains HTML parsing errors), `has_iframe` (true if an `<iframe>` tag is in the page document), `has_form_with_url` (true if a `<form>` element exists with a valid, static `src` attribute). References are added for `<a>` elements with valid (i.e. statically known and leading to a valid HTTP or HTTPS URL after resolution) `href` attributes, `<form>` elements with valid `action` attributes, and `<iframe>` elements with valid `src` attributes	`is_cert_valid` (false if expired or rejected by `rustls`), `cert_country`, `cert_reliability` (computed using the duration of the certificate and whether its issuer is trusted), `has_whois` (false if WHOIS could not be resolved for the domain), `domain_age` (in seconds, between the last update date and the domain registry expiry date), `domain_end_period` (in, seconds between the date of the extraction and the domain registry expiry date)

the prediction (benign or malicious) back to the client. However, if the domain is not found in the cache, the following steps are executed: (1) the URL is appended to the crawler's queue. (2) The crawler retrieves the hyperlink graph structure of the webpage, along with lexical and content features, and stores them in `CSV` format. (3) A trained phishGNN model is invoked to make an inference on the new domain, using the graph and features collected by the crawler as input. (4) The prediction is then sent back to the client and stored in the cache memory.

3.2 Web plugin/proxy

We implemented the proxy as a web plugin. Web plugins offer numerous benefits. They provide customization and enhanced functionality, allowing users to tailor their browsing experience. Additionally, they support accessibility, synchronize across platforms, and when used responsibly, significantly enhance the overall

browsing experience. Hence, our approach is completely transparent from the final user's perspective. We used Javascript for the development of the web plugin. Currently, we have only created a Mozilla Firefox compatible version[13]. Once the plugin intercepts the HTTP request of the client, it acts as an HTTP client towards the detection responder (which runs an HTTP server). Hence, the plugin sends an HTTP request containing the URL and waits for a boolean response. If the awaited response stands for a positive detection of a phishing website, then the plugin sends a warning to the user and blocks the HTTP request. However, if the awaited response stands for a negative detection, the plugin forwards the HTTP request made by the web client as it is to the corresponding web server. Unlike the existing commercial solutions that rely on crowd-sourced blacklists, it fully relies on PhishGNN detection engine.

Furthermore, we implemented a caching system within the extension using Firefox's *"localstorage"*[14]. e.g., if a domain is identified as non-phishing, it is cached, and subsequently, the extension refrains from intervention when the user accesses this domain in the future. This caching mechanism significantly reduces the number of requests sent.

3.3 The communication protocol

During the design of our end-to-end architecture, we faced the decision between two communication protocols, the HyperText Transfer Protocol (HTTP) and Message Queuing Telemetry Transport (MQTT), each of which bears its own merits. Indeed, MQTT is a reliable and fast protocol, consuming minimal bandwidth, a crucial characteristic for our extension, given its frequent requests to the remote server. With a lightweight header of only 2 bytes, our messages remain concise. The payload during client-to-server communication consists of an URL, while server-to-client responses typically convey a binary value of 0 or 1.

However, MQTT being a publish/subscribe protocol, it does not support a request/response system as such. Hence, it needs to implement an additional MQTT broker to communicate with the subscribers.

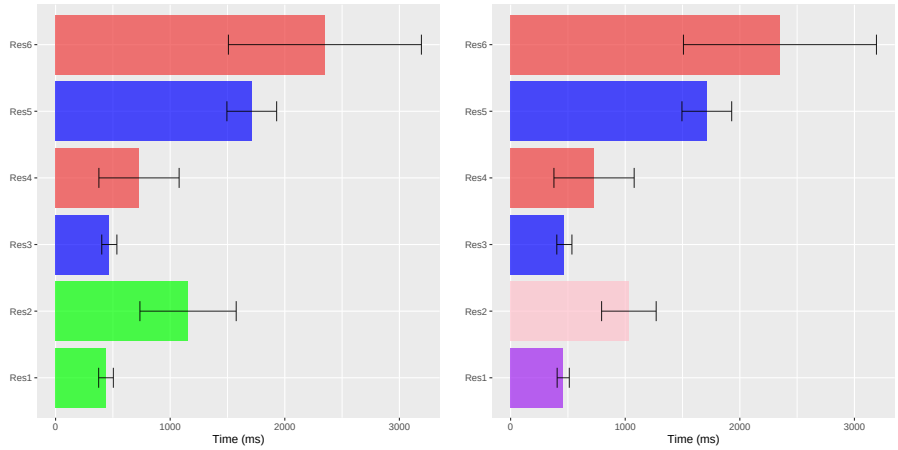
Consequently, the use of MQTT solution adds complexity to the end-to-end architecture which leads to additional delays for the end user. For these reasons, we chose to use HTTP. Hence, the detection responder (implemented in C language) implements an HTTP server that (1) waits for HTTP requests from the clients. Then, (2) triggers the detection engine, and (3) responds to the clients via an HTTP reply.

4 Performance evaluation and discussion

In our previous work [5] we showed how our detection engine PhishGNN outperforms the existing works. Hence, in this evaluation we focus on the performance of the end-to-end approach and the impact on the users' experience.

[13] The source code of the plugin is available on: `https://github.com/STERN3L/Semester_III_Project-Web_Plugin`

[14] https://developer.mozilla.org/fr/docs/Mozilla/Add-ons/WebExtensions/API/storage/local



(a) (b)

Fig. 5: Webpage loading times. Res3: time needed to load a benign website with our solution (DN in the cache); Res4: time needed to detect and block a phishing website with our solution (DN in the cache); Res5: time needed to load a benign website with our solution (DN not in the cache); Res6: time needed to detect and block a phishing website with our solution (DN not in the cache). (a) Res1: time needed to load a benign website without protection; Res2: time needed to load a phishing website without protection. (b) Res1: time needed to load a benign website using the Avast plugin; Res2: time needed to detect and block a phishing website using the Avast plugin.

For the evaluation of our approach, we implemented the architecture using Firefox in headless mode as client. We implemented the detection responder on an Intel Xeon Silver 4108 CPU @ 1.80 GHz with 45GB RAM and running CentOS Linux 7 64 bits. The Figure 5.a illustrates the webpage loading times acquired during the evaluation of our detection architecture. Each value on the plot corresponds to the average value across 10 tests.

Without our architecture, the time needed to load a benign website is in average 440 ms, while the time needed to load a phishing website[15] is around 1156 ms with a standard deviation of 420 ms. We attribute this distinction to the following reasons: (1) phishing websites lack optimization, (2) certain ones execute malicious JavaScript code, and (3) they are hosted on servers/platforms with inferior performance compared to authentic commercial websites. When using our approach we face three use-cases: *(1) The requested domain name is in the cache of the detection responder:* in this case, if the requested website is benign, the average time needed to load the website is around 469 ms. However, if the requested website is a phishing one, the time needed for the plugin (proxy) to get a response and stop the process is around 728 ms with a standard deviation of 350 ms. *(2) The requested domain name is not in the cache of the detection responder:* in this case, if the requested website is benign, the average time needed to load the website (after triggering a detection cycle) is in average

[15] The phishing websites used during our experimentations were obtained from the Phishtank list.

1712 ms with a standard deviation of 217 ms. However, if the requested website is a phishing one, the time needed for the plugin to get a response and stop the process is around 2350 ms with a standard deviation of 842 ms. While the duration remains acceptable to end-users, it is crucial to note that this occurs only when the domain name is not stored in the cache of the detection responder, which is shared by multiple users. *(3) The requested domain name is in the cache of the client:* (thanks to *localstorage* function in our implementation). In this case, the time needed for the whole process is just few microseconds.

To better understand the performances of our approach regarding the execution time, we compare it to one of the most used solutions, the Avast Online Security browser extension. The Figure 5.b shows the webpage loading times when Avast Online Security plugin is implemented on Firefox. We can observe that our approach performs better than the Avast solution when the domain name is in the cache of the detection responder (728 ms against 1032 ms to detect and block a phishing website). However, it requires more time when the domain name is not in the cache. We recall that the Avast approach relies on crowd-sourced blacklists that are often flawed as it was discussed earlier in this paper and does not have its own detection engine.

5 Conclusion and future works

In this paper, we have tackled the problem of phishing attacks and proposed an end-to-end detection approach that relies on Graph Neural Networks (GNNs). As far as we know, PhishGNN represents the first application of a GNN to the hyperlink structure of websites for the task of phishing detection. Furthermore, we provided a turnkey solution for the end-users in the form of a web browser extension. The evaluation of our approach, shows its efficiency and performance towards the end-users.

The Time To Live (TTL) setting plays a significant role in the context of our proposal and influence the balance between performance and data freshness, where caching is crucial to overall performance. Therefore, for our short-term future works, we plan to conduct a study/measurement campaign to determine the optimal values for this parameter. Afterwards, we aim to incorporate privacy support for end-users. Indeed, in addition to employing hashing techniques to conceal client-requested domain names as used by Google Safe Browsing, we intend to implement bloom filters which will enable clients to download compressed lists of domain names. Furthermore, we plan to develop versions of our client-side web browser plugin for the different existing web browsers.

References

1. 2023 Data Breach Investigations Report (DBIR). Technical report, Verizon, 2023.
2. 2023 State of the Phish. An in-depth exploation of user awareness, vulnerability and resilience. Technical report, Proofpoint, 2023.

3. Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. Malicious URL detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*, 2017.
4. Simon Bell and Peter Komisarczuk. An analysis of phishing blacklists: Google safe browsing, openphish, and phishtank. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–11, 2020.
5. Tristan Bilot, Grégoire Geis, and Badis Hammi. Phishgnn: A phishing website detection framework using graph neural networks. In *Proceedings of the 19th International Conference on Security and Cryptography*, volume 1, 2022.
6. Neil Chou Robert Ledesma Yuka Teraguchi and John C Mitchell. Client-side defense against web-based identity theft. *Computer Science Department, Stanford University*, 2004.
7. Gunikhan Sonowal and KS Kuppusamy. PhiDMA–A phishing detection model with multi-filter approach. *Journal of King Saud University-Computer and Information Sciences*, 32(1):99–112, 2020.
8. Victor E Adeyemo, Abdullateef O Balogun, Hammed A Mojeed, Noah O Akande, and Kayode S Adewole. Ensemble-based logistic model trees for website phishing detection. In *International Conference on Advances in Cyber Security*, pages 627–641. Springer, 2020.
9. Eduardo Benavides, Walter Fuertes, Sandra Sanchez, and Manuel Sanchez. Classification of phishing attack solutions by employing deep learning techniques: A systematic literature review. *Developments and advances in defense and security*, pages 51–64, 2020.
10. Joshua Saxe and Konstantin Berlin. eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. *arXiv preprint arXiv:1702.08568*, 2017.
11. Hung Le, Quang Pham, Doyen Sahoo, and Steven CH Hoi. URLNet: Learning a URL representation with deep learning for malicious URL detection. *arXiv preprint arXiv:1802.03162*, 2018.
12. Linshu Ouyang and Yongzheng Zhang. 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, 2021.
13. Shuaicong Yu, Changqing An, Tao Yu, Ziyi Zhao, Tianshu Li, and Jilong Wang. Phishing Detection Based on Multi-Feature Neural Network. In *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 73–79. IEEE, 2022.
14. Huiping Yao and Dongwan Shin. Towards preventing qr code based attacks on android phone using security warnings. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 341–346, 2013.
15. Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. Discovering browser extensions via web accessible resources. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 329–336, 2017.