Detecting Malicious Websites Based on JavaScript Content Analysis

Muhammad Fakhrur Rozi[1,2,a)] Tao Ban[1] Sangwook Kim[2] Seiichi Ozawa[2,3]
Takeshi Takahashi[1] Daisuke Inoue[1]

Abstract: Analyzing JavaScript contents has been a promising way to detect malicious websites. However, attackers often put malicious scripts in unreachable places and use complicated obfuscation tools to hinder the detection. Therefore, finding malicious scripts exhaustively inside the website tend to be time-consuming and ineffective in improving detection performance. To address these challenges, we introduce a novel approach to detecting malicious websites by analyzing the collective representation of the stack of JavaScripts in a website. First, we build a collective graph representation of a website by aggregating abstract syntax trees of all JavaScripts therein. Then, we use graph2vec to encode the graph into a vectorial representation. Finally, machine learning based detection is performed for identifying potentially harmful websites. Results showed that the proposed approach achieves high accuracy on a real-world dataset, outperforming prior approaches. We believe the result in this paper can open new opportunities for more effective malicious-website detection.

Keywords: Web-based attack, Machine Learning, Malicious JavaScript detection, JavaScript, graph2vec, Representation learning

1. Introduction

The malicious website has been a severe threat to Internet users. Cyber attackers spread their payload through malicious URLs embedded in buttons, text, images, or icons. Many types attacks of can be done when we accidentally access malicious websites, such as cross-site scripting (XSS) [31], drive-by-download [6], or injection attack [1]. They often use malicious JavaScript code or other suspicious code that brings users to get viruses, worms, trojan, or for the worst case their sensitive data.

There are still plenty of challenges regarding Internet users' protection from cyber threats. Attackers are improving their skill to evade the scanners by using some techniques such as obfuscation. They also hide the payload in unreachable places that can affect the performance of the detection system. Therefore, we need to analyze the target JavaScript's content to find characteristics of a malicious website instead of finding the actual payload which will be more time-consuming and ineffective.

In this research, we proposed a novel approach that analyzes websites' content based on the collection of JavaScript files. We capture the semantic meaning of the files that represents all actions of websites. We use the abstract syntax tree (AST) feature to extract the JavaScript program as the primary representation and encode them using a graph embedding model. We choose graph2vec [19] as an embedding model that can capture the structured and symbolic feature of the AST graph and yield a low-dimensional vectorial representation.

In summary, our contributions in this paper are as follow:

- We present a novel approach to detect malicious websites based on the collection of JavaScript files. The aggregation of all JavaScript information can represent the content of the websites so that we can get the signature of the malicious website for our detection system.
- We use AST as the feature of JavaScript files, which represent the structure of the program. We show that the AST feature effectively avoids obfuscation techniques that most attackers and normal programmers use to protect the code from reverse engineering.
- We carry the graph2vec model to transform the AST graph into an embedding vector representation. The graph2vec model captures the structure and the node feature of the AST graph as the characteristic of JavaScript, yielding a good representation of the JavaScript's content.

2. Related Works

Malicious website detection. Many researchers have researched malicious website detection with many approaches to get the best performance and fit into the current

[1] National Institute of Information and Communications Technology
[2] Graduate School of Engineering, Kobe University
[3] Center for Mathematical and Data Sciences, Kobe University
[a)] fakhrurrozi95@nict.go.jp

© 2021 Information Processing Society of Japan

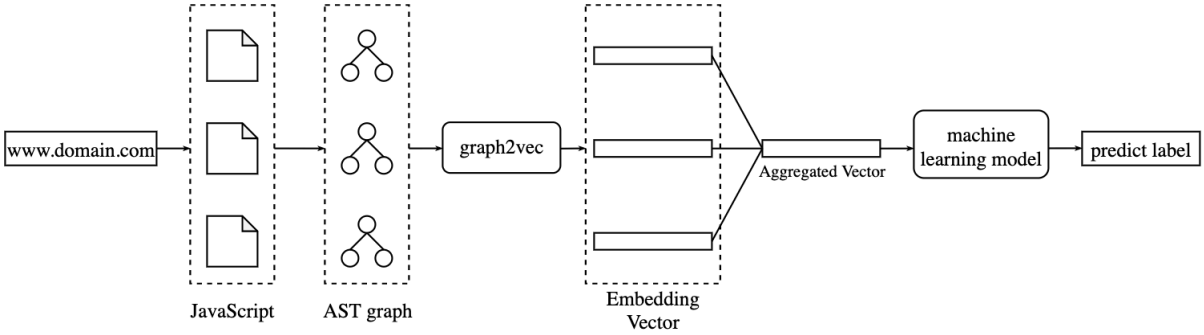

Fig. 1 The whole process of our proposed approach from a single website into prediction label.

condition. From the simplest approach, blocklisting method [25] is used by many anti-viruses scanners to store all recognizable websites, and we can utilize it in a detection system. This method still exists in many scanners as a conventional solution to protect the system [10][21]. However, if there are minor differences in the block list, the malicious data will be undetected. Therefore, research on website content using machine learning has been developed. Lexical features [13][15][16], host-based features [17][4][20], and content features such as HTML [7][2] and JavaScript [5][23] as the input of the learning models are used in these methods. The model can also vary from simple (ex. decision tree, Naive Bayes) to more complex (ex. deep learning, RNN, CNN) ones.

3. Methodology

3.1 Overview

Our goal is malicious websites detection. We define $W = \{w_1, w_2, ..., w_N\}$ as a set of websites that satisfies two conditions: accessible and containing JavaScript files. Given a domain name, we collect JavaScript files inside that website for every $w_i \in W$ in our dataset by crawling the related URLs until a certain level. Formally we can define the collection of JavaScript files for each website as $w_i = \{J_1^i, J_2^i, ..., J_j^i\}$ where J_j^i is the JavaScript file collected from a website. The size of $|w_i|$ is the number of JavaScript files that we can collect. We can parse each $J_j^i \in w_i$ into an AST graph representing how the program is written using a AST parser. During a pre-trained phase, we encode the AST graph into vectorial representation using the graph2vec model. After we aggregate by using `AVERAGE()` function for all representations to become a single representation, a website will have a numerical representation written as $X = \{x_1, x_2, ..., x_N | x_i \in \mathbb{R}^F\}$ where F is the dimension of the vector. Then, we use those representations as the input of the machine learning model to build a detection system for malicious websites. Figure 1 illustrates the whole process from a single website into a prediction label.

3.2 Collecting JavaScript Content

JavaScript is a well-known programming language for front-end development. That is one reason JavaScript is the key to a website that controls any logical actions. Therefore, the first step in our proposed approach is collecting as many JavaScript contents from a website. We assume that all files are well-represented that describes the collection of actions of the website.

To get all possible JavaScript files from a website, we build a web crawler to access all possible pages start from the main page. We define level of crawling where the first landing page is level-0 and the level is increased over the depth of crawling. We collect all JavaScript files inside the pages from level 0 until the maximum level that where we can still collect JavaScript's content. Figure 2 illustrates how we collect JavaScript files for each website in our dataset. We gather all JavaScript without any additional treatment for JavaScript files at every level. We believe that all actions running by JavaScript are connected that represent the whole action of the website. We also do not omit redundant files on the website because the repetition action is also suspicious or has a specific meaning.

3.3 Generating AST graph

In this research, we use the AST feature to represent the abstraction of the JavaScript program. After collecting all JavaScript files from the website using a crawler, we parse the JavaScript into a structural representation in tree graph form. The AST represents the abstract syntactic structure of the source code that covers any logic and statements. AST is widely used in compilers to represent the program code. Besides that, parsing the source code into a tree representation also can help the static analysis process, and we can avoid the obfuscated techniques.

AST consists of 69 syntactic units/tokens, including all statements and logical action in a JavaScript program. For example, function expressions, array expressions, Do-While statements, and many more [8]. Syntax tree format is derived from the original version of Mozilla Parser API [18], which is then formalized and expanded as the ESTree [9] specification. The AST graph consists of nodes and edges where nodes represent the syntactic units and edges represent the hierarchical relationship between two units. Figure 3 depicts how we transform a simple JavaScript statement

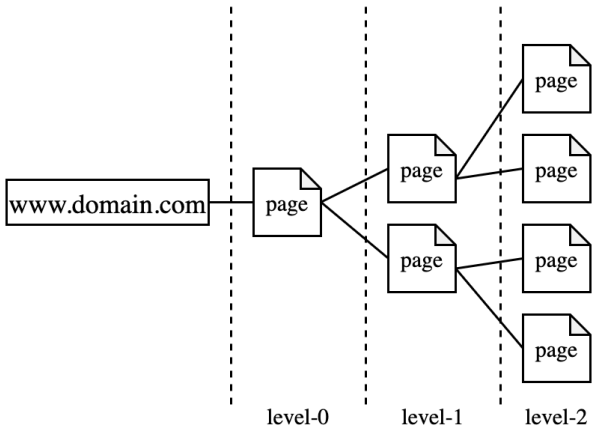

Fig. 2 Mechanism for collecting JavaScript files.

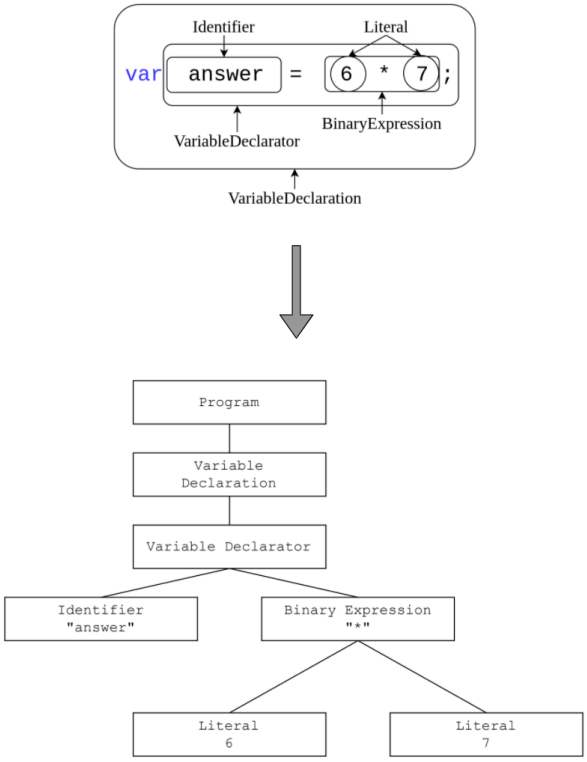

Fig. 3 Transforming a simple JavaScript statement into AST representation.

into an AST graph. The output is a tree representation that captures the structure of the JavaScript program. We do this process for all JavaScript collected before aggregating them into a single representation for each website.

3.4 JavaScript Content Representation

Given a set of graphs $\mathbb{G} = \{G_1, G_2, ..., G_N\}$ after transforming all JavaScript files into AST graph representation. Our goal is to encode the AST structure to become a vector representation for each file that can represent well the characteristics of the AST graph. We implement an unsupervised learning model to create an embedding vector for all AST graphs into a particular dimensional space. So, we can have a vocabulary embedding vector that store all vector representation that we can use to build a detection model.

There are some existing methods for graph embedding, such as Node2vec [11], LINE [26], DeepWalk [3], or GL2vec [27]. Some of them are node-based embedding models, and others are for the whole graph embedding model. In this research, we use a graph2vec model as an embedding method [19]. The graph2vec complements previous graph kernel methods that are lacking generalization. This model has exactly a similar concept with the doc2vec model [14] in the natural language processing field. The difference is just the object of the input. If we work on doc2vec, we can consider that a document is composed of multiple words that become sentences and paragraphs. However, in graph embedding cases, any graph is composed of atomic nodes connected with a set of edges. Besides that, we can consider the rooted graph as the atomic element of a graph. The rooted graph is considered better than just a single node because it encompasses higher-order neighborhoods, offering a richer representation of the graphs' composition and better captures the inherent non-linearity in the graphs.

Formally, let $G = (N, E, f)$ where N is a set of node, E is a set of edges, and f is a function to map each node with label l, then a rooted subgraph can be written as $sg = (N_{sg}, E_{sg}, f_{sg})$. In a given graph G_i, a rooted subgraph of degree D around node $n \in N$ contains all the reachable nodes in d hops from n. We follow the Weisfeiler-Lehman (WL) relabeling process [24] to extract subgraphs [32]. When $d > 0$, we get all the neighbors of $n \in N$ using the breadth-first algorithm. For each neighboring node n, we get its degree $d-1$ subgraph and save it in the same list. We will get the degree $d-1$ subgraph around the root node n and concatenate the same with a sorted list to obtain the intended subgraph $sg_n^{(d)}$.

After we get all rooted subgraphs, we can start to train the unsupervised model likes doc2vec to get the embedding vector representation of each AST graph. Similarly, given a set of AST graphs $\mathbb{G} = \{G_1, G_2, ..., G_N\}$ and a sequence of rooted subgraphs $SG = \{sg_1, sg_2, ..., sg_N\}$ sampled from AST graph $G_i \in \mathbb{G}$, graph2vec skip-gram learns a F dimensional embeddings of the graph $G_i \in \mathbb{G}$ and each rooted subgraph sg_i sampled from SG. The model works by considering a rooted subgraph $sg_i \in SG$ to be occurring in the context of graph G_i and tries to maximize the following log-likelihood:

$$\sum_{j=1} logPr(sg_j|G_i), \tag{1}$$

where the probability of $Pr(sg_j|G)$ is defined as,

$$\frac{exp(G \cdot sg_j)}{\sum_{sg \in \mathbf{V}} exp(G \cdot sg)}. \tag{2}$$

Here $\mathbf{V}$ is the vocabulary of all the rooted subgraph across all graph in G.

4. Experiments

4.1 Setup

Dataset. We evaluated our proposed approach to detect

Table 1 The whole set of websites used for the experiment.

Data Set	Total Files	#Benign	#Malicious
Training	4,272	2,129	2,143
Testing	1,069	533	536
Total	5,341	2,662	2,679

Table 2 Parameter setup.

Parameter	Values
learning rate	0.025
dimension size	128
#epochs	10
Maximum Node	500
minimal graph features occurrences	5
aggregate function	AVERAGE()

the maliciousness of websites by using a real-world dataset. We used the WarpDrive project dataset [30], which is the research project about the countermeasure of web-based attacks. They collected URLs from users that browsed specific websites through a mobile phone or PC. Due to our research is using the JavaScript contents of a website, we filtered the dataset based on whether we can still access the websites or not and the collectability of JavaScript files. We omitted websites that do not contain JavaScript information, which is not in our research scope. The overall summary of the number of malicious and benign websites we use in our experiments is described in Table 1.

Label aggregation. We labeled our dataset using VirusTotal [29], which analyzes each website as the ground truth for our experiment. Since the result of VirusTotal is the collection of multiple engine's labels, we have to define how we interpret the VirusTotal's result so we can have a single label based on the aggregation of all engine's labels. For this research, we considered a website with at least one malicious/suspicious label from any engine a detection target. Even though the malicious label is from the not popular engine and most of the popular ones give a safe label, there is still a probability of false-negative and the hazard flip (flip its label and then change it back within a day) [33] in the VirusTotal's result. The flip in VirusTotal may happen because the engine is not up-to-date for a specific type of malicious website that affects the result. For instance, phishing websites are more difficult to be analyzed because they can perform like an actual website.

Parameter setup. We implemented our proposed approach using some parameter setup to get the optimum result. For the pre-trained phase, we used the graph2vec model that is using the same way with doc2vec. So, we set the parameter using default values based on the KarateClub framework [12], such as learning rate is 0.025, the number of epochs is 10, the dimension size is 128, or the minimal count of graph features occurrences is 5. More details related to graph2vec model parameter setup can be seen in Table 2.

Moreover, we set the optimum parameters for all machine learning models by validating all combination possibilities and get the best performance. We tried to use various machine learning models to investigate the suitable model that can be used to get the best result. Our experiments considered some learning approaches, such as Naive Bayes, decision tree, XGBoost, logistic regression, simple neural network (multilayer perceptron), and support vector machine. Furthermore, to validate the performance for each model, we used 5-fold cross-validation.

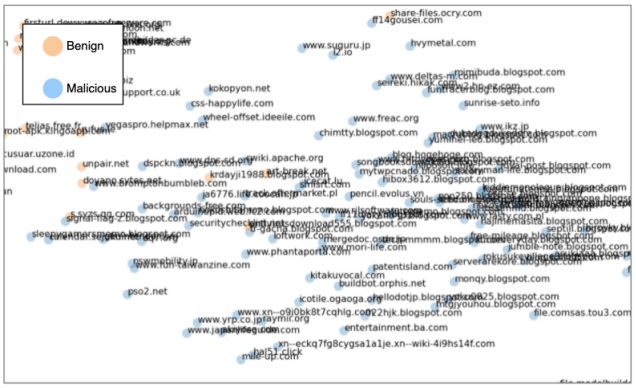

Fig. 4 Websites that has similar content.

4.2 Performance Evaluation

Pre-trained results. Figure 5 shows the scatter plot of the embedding vector for all websites in our dataset. The original dimension size of the embedding vector is 128. However, to show the distribution of the dataset, we used a t-distributed stochastic neighborhood (t-SNE) plot [28] to create a 2-dimensional representation. Even though the plot is inaccurate to represent the high dimension vector, we can still find a rough difference between malicious and benign samples. As we can see, that graph2vec model can extract the semantic information of websites based on the AST feature of all JavaScript contents inside them. We can get the content representation embedded in a low-dimensional vector by aggregating all representations into a single vector for each website. The closer distance of two dots in that plot indicates that two websites have similar contents in terms of JavaScript files and may have a similar purpose.

Detection results. Table 3 reports the performance of all machine learning models by using the result of the content representation of the website in the pre-trained phase. Because we already have the more straightforward representation, we can easily use that vector representation as the input of the learning model. However, building the model on such representation is still challenging because many websites are similar malicious and benign based on JavaScript contents. The performance result shows that the linear machine learning models, such as Naive Bayes, decision tree, XGboost, and logistic regression, have a lower detection performance than SVM with radial basis function (RBF) and a neural network.

5. Discussion

Content representation. Our approach exploits all information related to JavaScript files, which is the website's core, to perform a malicious action. We utilized the AST feature of JavaScript to get a vectorial representation that can be used for other processes. We try to capture

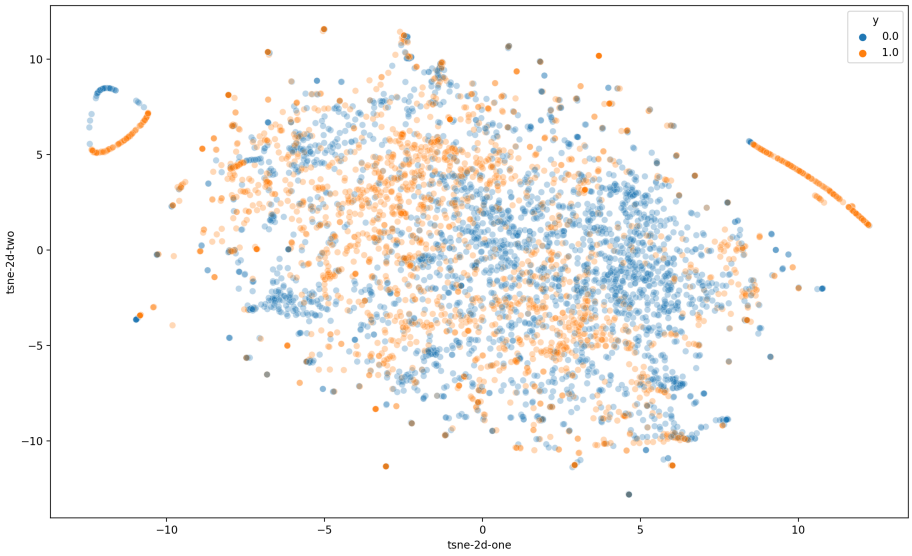

Fig. 5 t-SNE plot of embedding vector representation for all dataset.

Table 3 Performance of various models which were evaluated for this work

Model	Precision (%)	Recall (%)	F1-score(%)
Naive Bayes	67.13	64.16	62.42
Decision Tree	77.77	77.78	77.77
Logistic Regression	90.66	90.60	90.61
XGBoost	90.90	90.94	90.90
SVM with RBF	**93.96**	**93.89**	**93.92**

the semantic meaning of AST and aggregate all information into a single representation that explains the JavaScript's content. Our experiments show that graph2vec effectively embed the AST information into a vectorial representation that we can use to build a machine learning model. Figure 4 shows the labeled plot of websites that have close distance in 2-dimensional space. Some blogging websites are gathered in a particular area, implying that each JavaScript file's aggregation of graph2vec representation can describe accurate website content.

Phishing sites. In our dataset, there are three different labels that VirusTotal use in their system: malignant, phishing, malware. Malignant sites contain exploits or other malignant artifacts, malware sites distribute malware, and phishing sites try to steal users' credentials. Among three of them, the phishing site is the most challenging one in that the content of a website is pretty similar to the target. Many malicious sites in our dataset look legitimate by using the minor modification of the target's name. Since our approach is a content-based analysis, the phishing site are more challenging to have a good representation so that machine learning models can detect the malicious websites with a good accuracy.

Limitations. Our method still has some limitations in detecting malicious websites. First, our method uses the graph2vec embedding method for representing JavaScript's AST. However, due to the similarity concept with doc2vec, graph2vec has some limitations that we can also befound in doc2vec. For example, doc2vec cannot handle unseen documents because it only has a fixed vocabulary embedding matrix that cannot be extended to embed new documents. That condition also applies to graph2vec. A new graph not included in a dataset does not have a vectorial representation, causing false positive or negative on the new dataset. To address this limitation, we can refer to [22] that they introduced a new model inspired by graph embedding and cross-lingual vector space representation technique to cover unseen data. Alternatively, we can use a simple strategy to find the most similar representation to define the unseen data. It only works if the number of unseen data is pretty small, otherwise it will influence the result significantly.

6. Conclusions

In this research, we present a novel approach to detect malicious websites based on the website's content using JavaScript feature information. Our approach tries to use the collection of JavaScript files inside a website, and we transform them to AST graph representations. We use graph2vec to encode each AST graph to become embedding vector representations. By aggregating all vectors within the same website, we can build a machine learning model for malicious website detection. Experimental results show

that our proposed approach is practical to detect whether a website is malicious or benign. For future works, we consider the limitation of the graph2vec model that cannot handle unseen data, and then we will find a layer-wise embedding model that can be used for unseen data. Enriching a website's content feature is needed to improve the performance, especially when handling phishing websites.

Acknowledgement

This research was partially supported by the Ministry of Education, Science, Sports, and Culture, Grant-in-Aid for Scientific Research (B) 21H03444.

References

[1] Boyd, S. W. and Keromytis, A. D.: SQLrand: Preventing SQL Injection Attacks, *Applied Cryptography and Network Security* (Jakobsson, M., Yung, M. and Zhou, J., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 292–302 (2004).

[2] Canali, D., Cova, M., Vigna, G. and Kruegel, C.: Prophiler: A Fast Filter for the Large-Scale Detection of Malicious Web Pages, *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, New York, NY, USA, Association for Computing Machinery, p. 197–206 (online), DOI: 10.1145/1963405.1963436 (2011).

[3] Chanpuriya, S., Musco, C., Sotiropoulos, K. and Tsourakakis, C. E.: DeepWalking Backwards: From Embeddings Back to Graphs, *CoRR*, Vol. abs/2102.08532 (online), available from ⟨https://arxiv.org/abs/2102.08532⟩ (2021).

[4] Choi, H., Zhu, B. B. and Lee, H.: Detecting Malicious Web Links and Identifying Their Attack Types, *Proceedings of the 2nd USENIX Conference on Web Application Development*, WebApps'11, USA, USENIX Association, p. 11 (2011).

[5] Choi, Y., Kim, T., Choi, S. and Lee, C.: Automatic Detection for JavaScript Obfuscation Attacks in Web Pages through String Pattern Analysis, *Future Generation Information Technology* (Lee, Y.-h., Kim, T.-h., Fang, W.-c. and Sezak, Dominik, eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 160–172 (2009).

[6] Cova, M., Kruegel, C. and Vigna, G.: Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code, *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, New York, NY, USA, Association for Computing Machinery, p. 281–290 (online), DOI: 10.1145/1772690.1772720 (2010).

[7] Desai, A., Jatakia, J., Naik, R. and Raul, N.: Malicious web content detection using machine leaning, *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pp. 1432–1436 (online), DOI: 10.1109/RTEICT.2017.8256834 (2017).

[8] Esprima: Syntax Tree Format, https://docs.esprima.org/en/latest/syntax-tree-format.html.

[9] ESTree: The ESTree Spec, https://github.com/estree/estree.

[10] Felegyhazi, M., Kreibich, C. and Paxson, V.: On the Potential of Proactive Domain Blacklisting, *Proceedings of the 3rd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'10, USA, USENIX Association, p. 6 (2010).

[11] Grover, A. and Leskovec, J.: node2vec: Scalable Feature Learning for Networks, *CoRR*, Vol. abs/1607.00653 (online), available from ⟨http://arxiv.org/abs/1607.00653⟩ (2016).

[12] KarateCLub: KarateClub, https://karateclub.readthedocs.io/en/latest/index.html.

[13] Kolari, P., Finin, T. and Joshi, A.: SVMs for the Blogosphere: Blog Identification and Splog Detection, pp. 92–99 (2006).

[14] Le, Q. V. and Mikolov, T.: Distributed Representations of Sentences and Documents, *CoRR*, Vol. abs/1405.4053 (online), available from ⟨http://arxiv.org/abs/1405.4053⟩ (2014).

[15] Ma, J., Saul, L. K., Savage, S. and Voelker, G. M.: Beyond blacklists: learning to detect malicious web sites from suspicious URLs, *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1245–1254 (2009).

[16] Ma, J., Saul, L. K., Savage, S. and Voelker, G. M.: Identifying Suspicious URLs: An Application of Large-Scale Online Learning, *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, New York, NY, USA, Association for Computing Machinery, p. 681–688 (online), DOI: 10.1145/1553374.1553462 (2009).

[17] McGrath, D. K. and Gupta, M.: Behind Phishing: An Examination of Phisher Modi Operandi, *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, USA, USENIX Association (2008).

[18] Mozilla: Mozilla Parser API, https://wiki.mozilla.org/JavaScript:-SpiderMonkey:Parser_API.

[19] Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y. and Jaiswal, S.: graph2vec: Learning Distributed Representations of Graphs, *ArXiv*, Vol. abs/1707.05005 (2017).

[20] Passerini, E., Paleari, R., Martignoni, L. and Bruschi, D.: FluXOR: Detecting and Monitoring Fast-Flux Service Networks, *Detection of Intrusions and Malware, and Vulnerability Assessment* (Zamboni, D., ed.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 186–206 (2008).

[21] Prakash, P., Kumar, M., Kompella, R. R. and Gupta, M.: PhishNet: Predictive Blacklisting to Detect Phishing Attacks, *2010 Proceedings IEEE INFOCOM*, pp. 1–5 (online), DOI: 10.1109/INFCOM.2010.5462216 (2010).

[22] Prokhorov, V., Pilehvar, M. T., Kartsaklis, D., Lio, P. and Collier, N.: Unseen Word Representation by Aligning Heterogeneous Lexical Semantic Spaces, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, No. 01, pp. 6900–6907 (online), DOI: 10.1609/aaai.v33i01.33016900 (2019).

[23] Rozi, M. F., Kim, S. and Ozawa, S.: Deep Neural Networks for Malicious JavaScript Detection Using Bytecode Sequences, *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8 (online), DOI: 10.1109/IJCNN48605.2020.9207134 (2020).

[24] Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K. and Borgwardt, K. M.: Weisfeiler-Lehman Graph Kernels, *Journal of Machine Learning Research*, Vol. 12, No. 77, pp. 2539–2561 (online), available from ⟨http://jmlr.org/papers/v12/shervashidze11a.html⟩ (2011).

[25] Sinha, S., Bailey, M. and Jahanian, F.: Shades of grey: On the effectiveness of reputation-based "blacklists", *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 57–64 (online), DOI: 10.1109/MALWARE.2008.4690858 (2008).

[26] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. and Mei, Q.: LINE: Large-Scale Information Network Embedding, *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, Republic and Canton of Geneva, CHE, International World Wide Web Conferences Steering Committee, p. 1067–1077 (online), DOI: 10.1145/2736277.2741093 (2015).

[27] Tu, K., Li, J., Towsley, D., Braines, D. and Turner, L. D.: gl2vec: Learning Feature Representation Using Graphlets for Directed Networks, *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 216–221 (online), DOI: 10.1145/3341161.3342908 (2019).

[28] van der Maaten, L. and Hinton, G.: Visualizing Data using t-SNE, *Journal of Machine Learning Research*, Vol. 9, No. 86, pp. 2579–2605 (online), available from ⟨http://jmlr.org/papers/v9/vandermaaten08a.html⟩ (2008).

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

[30] WarpDrive: WarpDrive Project, https://warpdrive-project.jp/.

[31] Wassermann, G. and Su, Z.: Static detection of cross-site scripting vulnerabilities, *2008 ACM/IEEE 30th International Conference on Software Engineering*, pp. 171–180 (online), DOI: 10.1145/1368088.1368112 (2008).

[32] Yanardag, P. and Vishwanathan, S.: *Deep Graph Kernels*, p. 1365–1374 (online), available from ⟨https://doi.org/10.1145/2783258.2783417⟩, Association for Computing Machinery (2015).

[33] Zhu, S., Shi, J., Yang, L., Qin, B., Zhang, Z., Song, L. and Wang, G.: Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines, *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, pp. 2361–2378 (2020).