Smart Vulnerability Assessment for Scientific Cyberinfrastructure: An Unsupervised Graph Embedding Approach

Steven Ullman
Department of Management Information Systems
University of Arizona
Tucson, Arizona U.S.
stevenullman@email.arizona.edu

Sagar Samtani
Department of Operations and Decision Technologies
Indiana University
Bloomington, Indiana U.S.
ssamtani@iu.edu

Ben Lazarine
Department of Management Information Systems
University of Arizona
Tucson, Arizona U.S.
benlazarine@email.arizona.edu

Hongyi Zhu
Department of Information Systems and Cyber Security
UTSA
San Antonio, Texas U.S.
hongyi.zhu@utsa.edu

Benjamin Ampel
Department of Management Information Systems
University of Arizona
Tucson, Arizona 85721
bampel@email.arizona.edu

Mark Patton
Department of Management Information Systems
University of Arizona
Tucson, Arizona 85721
mpatton@email.arizona.edu

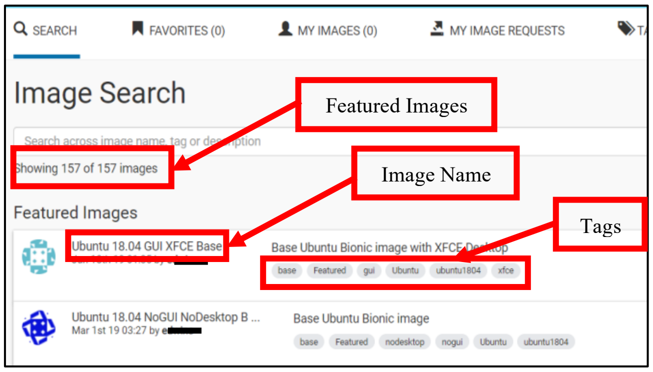
Hsinchun Chen
Department of Management Information Systems
University of Arizona
Tucson, Arizona 85721
hsinchun@email.arizona.edu

***Abstract*—The accelerated growth of computing technologies has provided interdisciplinary teams a platform for producing innovative research at an unprecedented speed. Advanced scientific cyberinfrastructures, in particular, provide data storage, applications, software, and other resources to facilitate the development of critical scientific discoveries. Users of these environments often rely on custom developed virtual machine (VM) images that are comprised of a diverse array of open source applications. These can include vulnerabilities undetectable by conventional vulnerability scanners. This research aims to identify the installed applications, their vulnerabilities, and how they vary across images in scientific cyberinfrastructure. We propose a novel unsupervised graph embedding framework that captures relationships between applications, as well as vulnerabilities identified on corresponding GitHub repositories. This embedding is used to cluster images with similar applications and vulnerabilities. We evaluate cluster quality using Silhouette, Calinski-Harabasz, and Davies-Bouldin indices, and application vulnerabilities through inspection of selected clusters. Results reveal that images pertaining to genomics research in our research testbed are at greater risk of high-severity shell spawning and data validation vulnerabilities.**

***Keywords*—Scientific cyberinfrastructure, vulnerability scanning, Graph Embedding, GitHub, virtual machine**

I. Introduction

The rapid advancement and development of computing technologies over the past decade has allowed for an unprecedented level of innovative scientific research. From DNA sequencing to the simulation of planetary formations and black hole imaging, interdisciplinary research teams have benefited tremendously from scalable, high performance computing environments. The National Science Foundation (NSF) has funded key Large Facilities (LFs), designated as scientific cyberinfrastructure (CI), to provide agile computing platforms to thousands of researchers [1]. Such NSF-funded installations include the Open Science Grid, TeraGrid, CyVerse, Jetstream, and Chameleon Cloud [2]–[4]. Users accessing these environments can develop their own custom-built virtual machine (VM) images to execute their desired scientific tasks. Figure 1 illustrates a subset of sample images that users can launch, e.g., *Ubuntu 18.04 GUI XFCE Base*.



Ubuntu 18.04 GUI XFCE Base [redacted]
Base Ubuntu Bionic image with XFCE Desktop
Ubuntu 18.04 NoGUI NoDesktop B ...
Mar 1st 19 03:27 by e[redacted]
Base Ubuntu Bionic image

Fig. 1. Examples of User-Developed VM Images

These images provide metadata such as image name, author, date published, description, and tags of keywords related to the image. In a typical workflow, users review names, tags and descriptions to identify images most suitable for their task. Once an image is selected, a user can spawn an instance of an arbitrary size, ranging from single to multiple core instances, containing varying levels of RAM. Users often install open source (e.g., GitHub) packages and applications on these images to help support their desired analytics.

Despite offering unprecedented convenience, scalability, and scientific efficiency, applications from public repositories can contain vulnerabilities undetectable by conventional scanners (e.g., Nessus) [5]–[7]. These issues can potentially disrupt high-impact scientific workflows if exploited.

In light of these significant ramifications, this research aims to identify the installed applications, their relationships to other applications, their vulnerabilities, and how they vary across images in scientific cyberinfrastructure. We develop a novel graph embedding approach that generates low-dimensional vector representations of each VM image based on installed applications, their dependencies, and vulnerabilities. Our results indicate that we can cluster images with similar applications, providing a more intelligent and targeted means for assessing and mitigating vulnerabilities.

The remainder of this paper is organized as follows. First, we review literature related to conventional and machine/deep learning-based device fingerprinting, and unsupervised graph embedding. Second, we present our research testbed and design, as well as our vulnerability assessment of the image applications within the scientific CI environment. Finally, we discuss our results and future direction for this research.

II. Literature Review

We review two key areas of literature. First, we review literature on conventional and machine/deep learning-based fingerprinting to identify prevailing techniques and data. Second, we review unsupervised graph embedding methods to identify how to generate low-dimensional representations from unlabeled graph-structured data.

A. *Conventional and Machine/Deep Learning-Based OS Fingerprinting*

Operating system (OS) fingerprinting is a commonly used method for identifying and representing devices on a network. Two fingerprinting approaches exist: passive and active. Passive approach identifies the device OS by observing network traffic, while active approach directly sends packets to a machine and analyzes the response [8]. These approaches are used to remotely gather OS data to generate an identifying signature for inventorying, updating, and/or patching outdated systems [9], [10], using tools such as Nmap, Ettercap, and p0f [11]. Generated fingerprints from these tools are useful for identifying device properties but do not allow for comprehensive direct comparisons between machines. We expand our review to gain insight from machine and deep learning-based fingerprinting techniques.

Device fingerprinting has seen widespread adoption of machine and deep learning techniques in recent years. These techniques have been popularized due to the exponentially increasing number of Internet of Things (IoT) devices. New methods have been developed to generate fingerprints for endpoint and device identification [12]–[14], and device localization/positioning [15]–[18]. Researchers have successfully leveraged data testbeds consisting of IoT devices, their network traffic data, and radio frequency data for input into machine/deep learning models [19].

While fingerprinting has been performed using a variety of features and sources, prevailing techniques omit host features such as installed applications and their dependencies [20]. These applications can include vulnerabilities linked to certain features (e.g. dependencies) [21]. In scientific CI, it is imperative to understand inter-application relationships within each VM image. This requires a method that captures application relationships to create a holistic representation.

B. *Unsupervised Graph Embedding Methods*

Graph analytics are a group of methods that can capture information that is hidden within graphs. Graph embedding methods are effective in generating fixed representations of entire graphs in a Euclidean space while still preserving the graph structures [22]. This representation can then be used for subsequent tasks, such as clustering or classification. There are four levels of granularity for graph embedding methods: node, edge, substructure, and whole graph embedding. Given our task of creating a representation of an entire image utilizing unlabeled data, we review relevant unsupervised whole-graph embedding methods.

Graph embedding methods can be split into two major categories based on their operations. Graph2vec and GL2vec are graph kernel-based, whereas NetLSD, GeoScattering, SF, and FGSD use spectral fingerprinting methods based on extracted graph statistics [23]. Inspired by doc2vec, graph2vec uses negative sampling to create rooted subgraphs of the nodes within a graph and trains a skip-gram model to maximize the probability of predicting subgraphs [24]. GL2vec is an extension of graph2vec that can handle edge labels [25]. NetLSD [26], GeoScattering [27], SF [28], and FGSD [29] all rely on statistical characteristics of the graph and spectral features. Kernel-based methods are often preferred as they create whole graph and feature embeddings jointly. Consequently, they are suitable for downstream clustering or classification algorithms.

Graph embedding methodology is similar to deep learning methods found in device fingerprinting, as it uses data features to create a single representation. Therefore, they can be leveraged to capture representations that provide more comprehensive views of a device or system.

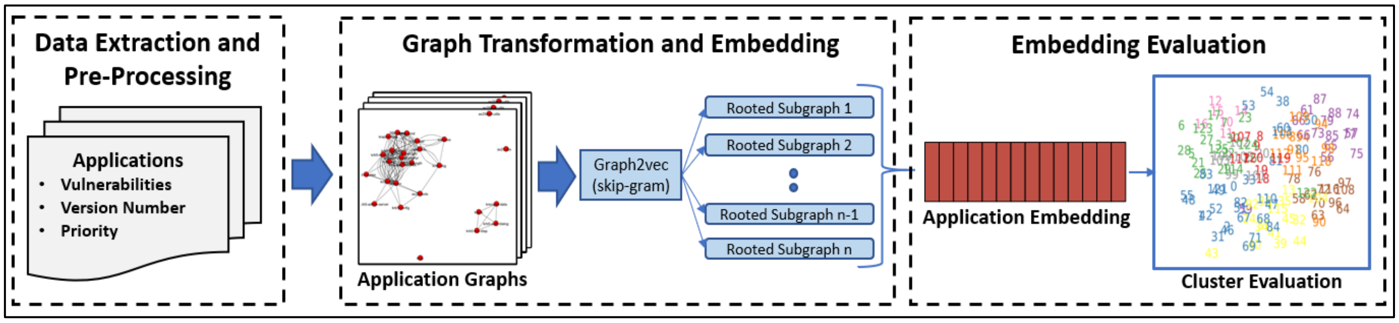


Fig. 2. Proposed Graph Embedding Framework for Images

III. Research Gaps and Questions

We have identified several limitations from previous literature. First, ML/DL-based fingerprinting has been used for downstream tasks such as device positioning yet omits vulnerability information. Second, prevailing fingerprinting techniques have primarily used temporal data and have not been applied on datasets to capture application relationships and their features. Third, graph embedding methods have not yet been formulated for system fingerprinting. These limitations motivate the following research questions:

1) How can we develop an unsupervised deep learning framework to automatically generate a representation of a VM image in scientific cyberinfrastructure?

2) How can we capture the image applications, their relationships, and detected vulnerabilities using a graph embedding?

IV. Research Design and Testbed

We propose a novel unsupervised graph embedding framework to answer these questions. The framework, shown in Figure 2, has three components: (1) Data Extraction and Pre-Processing; (2) Graph Transformation and Embedding; and (3) Embedding Evaluation. We describe each component in further detail in the following sub-sections.

A. *Data Extraction and Pre-Processing*

We collected user system data from a major NSF-funded scientific cyberinfrastructure platform with more than 7,000 participating institutions and 45,000 users in life sciences. To protect their privacy, we have anonymized their name. The data collection has two phases: extracting application data from the VM images and vulnerability assessment on the collected applications. We describe both in turn.

1) *Image Data Collection*

Using the Advanced Package Tool (APT), we collected software application data from each launchable image hosted on the CI platform, including current and previous versions of images. This resulted in 148 total images. Of these images, 126 were Ubuntu-based Linux distributions. Given the distribution of Linux variants, we use the subset of Ubuntu-based images. Over 6 million packages were collected from these images. Most applications provide a URL to the homepage where the application is maintained. We summarize the application homepage distribution in Table 1.

TABLE I. Application Homepages Per Image

Application Homepage	Number of Applications	Percent of Total Applications
GitHub	817,646	11.96%
Gnu	631,254	9.23%
Sourceforge	444,254	6.5%
Metacpan	257,842	3.77%
Haskell	158,013	2.31%
kde	151,451	2.21%
launchpad	145,707	2.13%
null	1,321,399	19.32%
Other	2,911,046	42.57%

The leading domain is GitHub, an open social coding repository, which more than 800,000 applications reference. 19.32% of our applications do not have homepages, and 42.57% lead to other homepages with less than 2% of total applications. Given GitHub's prevalence, we analyze the relevant GitHub repositories that maintain the collected applications. We summarize our findings in Table 2.

TABLE II. Application GitHub Repository Summary

Data Type	Root Repositories
Number of Repositories (distinct)	8,701
Number of Forks	1,258,075
Number of Commits	5,200,563
Size (files)	43,358,089
Number of Issues	225,327
Number of Languages	69
Top Programming Languages	Python (1,811), C (1,536)

We identified 8,701 distinct repositories containing over 43 million files from GitHub that are related to the applications found on collected images. Python and C are the most frequent, for approximately 40% of all repositories. GitHub vulnerability scanners are then assessed based on functionality with those programming languages.

2) *Application Vulnerability Assessment*

We reviewed 14 GitHub vulnerability scanners and selected two based on coverage and usage. Bandit and

FlawFinder are both scanners designed to identify secrets, insecurities, and attack vulnerabilities that exist within Python and C code hosted on GitHub. Both tools categorize vulnerabilities into High, Medium, and Low severities. High vulnerabilities include SSL with bad versions, blacklisted Python input calls, and deprecated libraries. Medium vulnerabilities include hardcoded SQL expressions, paramiko calls in Python, and various XML methods. Finally, Low vulnerabilities include try/except functions, blacklisted Python imports, and certain subprocess spawns. We scanned each repository in our testbed for vulnerabilities and summarize selected results in Table 3.

TABLE III. VULNERABILITY ASSESSMENT RESULTS FOR BANDIT

Vulnerable Applications	Vulnerabilities		Most Frequent Vulnerable Repository/Application	Number of Vulnerabilities
47,932	High	1,634	usit-gd/zabbix	104
			CoreSecurity/impacktct	77
			ctuning/ck	30
91,571	Medium	9,959	PacificBiosciences/pbcore	1,599
			annulen/webkit	373
			feist/pcs	293
144,481	Low	221,869	sympy/sympy	66,344
			annulen/webkit	9,559
			dask/dask	4,581

In total, 233,642 vulnerabilities were detected through Bandit, while 25,170 vulnerabilities were detected by FlawFinder. 1,634 vulnerabilities are identified as high severity, propagating across 47,932 applications for each image. In the medium severity results, both PacificBiosciences/pbcore and feist/pcs are directly linked to biology/health APIs. These tools report and detect individual vulnerabilities but do not provide a means to assess which images contain vulnerable applications or their dependencies. A different method is required to capture inter-application relationships that the vulnerability scanners overlook.

B. *Graph Transformation and Embedding*

A graph embedding-based approach can capture inter-application relationships and provide a fine-grained representation of the images for downstream clustering tasks. The installed applications on an image can be represented as a graph. Relationships are created between applications based on shared dependencies. Following this principle, we define our graphs as $G=(A,E,F)$, where G is an undirected graph, A is the node set, $\{u_1, u_2, u_3, \dots u_n\}$, of GitHub-maintained applications in an image, E is the edge set, $\{e_1, e_2, e_3, \dots e_n\}$, of all edges between applications based on shared dependencies, and F is a feature matrix of all vulnerabilities for that application.

As indicated in the literature review, selection of graph embedding algorithm is contingent upon the data characteristics, task, and research objective. The described graph formulation includes nodal features and undirected edges. Our proposed analytics requires a method that operates without prior knowledge of the graph and incorporates nodal features. Therefore, we select graph2vec. The embedding generation process with graph2vec follows five steps [24]:

- **Step 1:** Nodes are negatively sampled and relabeled to create rooted subgraphs in the graph.
- **Step 2:** A skip-gram model is trained to maximize the probability of predicting subgraphs that exist in the input graph.
- **Step 3:** The embedding is then learned from the extracted subgraphs over several epochs.
- **Step 4:** A final embedding is produced as a one-hot vector.
- **Step 5:** Steps 1-4 repeat for each graph in a given set.

In our case, subgraphs are generated around each application, capturing its dependencies with other applications. Images that have similar applications and dependencies will thus have similar embeddings.

C. *Embedding Evaluation*

The generated image embeddings are subsequently clustered using K-means, a prevailing partitional clustering algorithm. We evaluate these clusters using Silhouette (SI), Calinski-Harabasz (CH), and Davies-Bouldin (DBI) indices to help identify the optimal number of clusters by measuring cluster quality, maximizing intra-cluster similarities and minimizing inter-cluster differences. SI uses average dissimilarity between points to show the structure of the data and its possible clusters [30]. CH represents the ratio of within-cluster and between cluster dispersion, where a higher number represents well separated and compact clusters [31]. DBI measures the ratio of within-cluster to between-cluster distances [32]. Each metric has been used extensively in previous clustering research [33], [34].

V. RESULTS AND DISCUSSION

We evaluate for cluster sizes from 3 to 20 to identify the optimal number of clusters. SI scores closer to one, a DBI score closer to zero, and high CH ratio indicate stronger performance. We summarize evaluation results in Table 4. The best performance is highlighted in boldface.

TABLE IV. IMAGE CLUSTER EVALUATION METRICS

K Clusters	Evaluation Metrics		
	Silhouette	Calinski-Harabasz	Davies-Bouldin
3	0.429	58.526	1.116
6	0.709	158.762	0.625
9	**0.824**	415.824	**0.37**
10	0.808	466.881	0.398
20	0.622	**992.431**	0.440

Evaluation results indicate that nine-clusters provides the highest SI at 0.824, and lowest DBI score at 0.37, while still maintaining a high CH ratio. Given these results, we run k-means for nine clusters and plot the results in Figure 3. The numbers are the ID for each image. Clusters are color-coded, circled, and labeled.

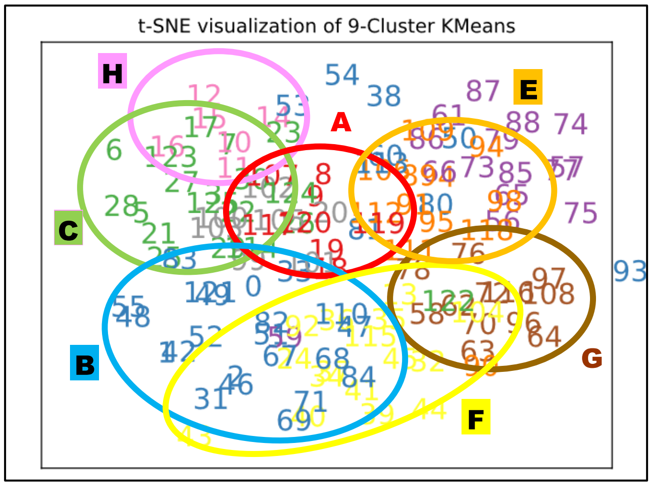


Fig. 3. Selected Image Clusters

The image clusters are relatively self-contained, with some overlap. Cluster B contains the highest number of images with 31, and cluster H contains the lowest number with six images. The average cluster size was 14 images. Cluster C contains images that are primarily base Ubuntu distributions. Cluster E contains images primarily loaded with RStudio. Clusters A, F, and G contain images designed for RNA and hybridized genome sequencing. We present the average number of vulnerabilities per cluster for each vulnerability in Figure 4.

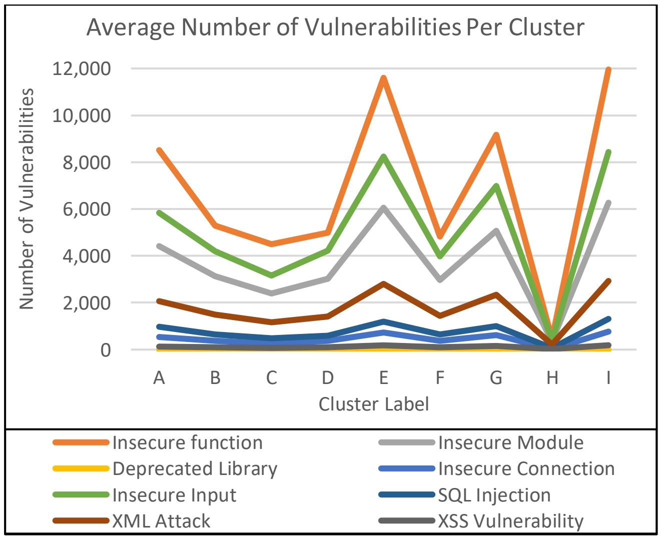


Fig. 4. Average Number of Vulnerabilities by Type Per Cluster

The most frequent type of vulnerability is related to insecure functions implemented in the code. This issue has over 8,000 counts in clusters A, E, G, and I. Clusters E and I possessed the highest number of average vulnerabilities. These clusters contained significantly higher counts of insecure function, input, and module vulnerabilities, at 11,954, 8,433, and 6,269, respectively. Cluster H contained the minimum, at 459, 394, and 350. Cluster H contained 1,500 total vulnerabilities, the lowest average number.

We further examine clusters H and I based on the difference in total number and severities of vulnerabilities. Cluster H contains six images with low severity vulnerabilities. 462 vulnerabilities were related to spawning subprocesses without shells and 210 vulnerabilities pertained to data validation. Cluster I contains seven images with high-severity insecure input and insecure function vulnerabilities that seldom occur in cluster H. These include 1,129 shell spawning issues and 612 data input functions. Vulnerabilities that spawn shells pose tremendous risk, as hackers can exploit these shells to execute arbitrary commands and disrupt operations. For example, *'rm -rf /'* can be passed as a parameter to a spawned shell to delete all files in the root directory. This destructive attack shell injection command that can set back scientific workflows by months if data and custom developed programs are erased. In cluster I, these specifically affect images that provide computational resources for genomics-related scientific workflows. Given the frequency and severity of these vulnerabilities, images in cluster I are at a much higher risk of disruption compared to those in cluster H.

These results suggest that images in clusters E and I should be prioritized for vulnerability mitigation, followed closely by those in clusters A and G. Scientific CI administrators can follow two strategies to remediate the detected vulnerabilities. First, issues should be opened on corresponding GitHub repositories to alert the maintainer of the specific insecure functions. Second, automated notifications can be sent to image users of the identified vulnerable applications and/or by specific vulnerability types. For shell spawning vulnerabilities, users should ensure that they change the shell parameter to 'False' within the related Python file. For input validation vulnerabilities, users should ensure that they properly sanitize their data, omitting potential arbitrary code prior to execution.

VI. Conclusion and Future Directions

Scientific CI provides environments to thousands of scientists that enable them to execute high-impact scientific inquiries and discovery. However, these environments may contain unconventional vulnerabilities, which expose users to potential disruption of high-impact scientific workflows. Our proposed research framework provides a novel approach for automatically detecting and grouping unconventional vulnerabilities applicable to multiple scientific CI. User images are grouped together based on similar applications and vulnerabilities. As a result, they can facilitate targeted mitigation and remediation activities.

There are several promising directions for future work. First, we intend to incorporate multiple data sources from other CI's to further demonstrate the generalizability of the proposed approach. Second, we plan to create a more holistic representation through a multi-view learning strategy incorporating additional image features. Both directions can further help improve scientific CI cybersecurity.

VII. Acknowledgements

This material is based upon work supported by the National Science Foundation under NSF grants: OAC-1917117 (CICI), DGE-1921485 (SFS), and CNS-1850362 (CRII and SaTC).

References

[1] E. Deelman et al., "Cyberinfrastructure center of excellence pilot: Connecting large facilities cyberinfrastructure," in Proceedings - IEEE 15th International Conference on eScience, eScience 2019, Sep. 2019, pp. 449–457.

[2] J. Mambretti, J. Chen, and F. Yeh, "Next generation clouds, the chameleon cloud testbed, and software defined networking (SDN)," in Proceedings - 2015 International Conference on Cloud Computing Research and Innovation, ICCCRI 2015, Feb. 2016, pp. 73–79.

[3] C. A. Stewart et al., "Jetstream: A self-provisioned, scalable science and engineering cloud environment," in ACM International Conference Proceeding Series, Jul. 2015, vol. 2015-July, pp. 1–8.

[4] C. A. Stewart, M. Link, S. Simms, D. Y. Hancock, and G. C. Fox, "What is Cyberinfrastructure?" 2010. Accessed: Jul. 22, 2020.

[5] M. Meli, M. R. Mcniece, and B. Reaves, "How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories" 2019.

[6] M. El, E. McMahon, S. Samtani, M. Patton, and H. Chen, "Benchmarking vulnerability scanners: An experiment on SCADA devices and scientific instruments," in 2017 IEEE International Conference on Intelligence and Security Informatics: Security and Big Data, ISI 2017, Aug. 2017, pp. 83–88, doi: 10.1109/ISI.2017.8004879.

[7] C. R. Harrell, M. Patton, H. Chen, and S. Samtani, "Vulnerability assessment, remediation, and automated reporting: Case studies of higher education institutions," in 2018 IEEE International Conference on Intelligence and Security Informatics, ISI 2018, Dec. 2018, pp. 148–153, doi: 10.1109/ISI.2018.8587380.

[8] F. Veysset, O. Courtay, and O. Heen, "New Tool and Technique for Remote Operating System Fingerprinting-Full Paper," 2002. Accessed: Jul. 20, 2020.

[9] L. Bilge and T. Dumitras, "Before we knew it: An empirical study of zero-day attacks in the real world," in Proceedings of the ACM Conference on Computer and Communications Security, 2012, pp. 833–844.

[10] W. A. H. M. Ghanem and B. Belaton, "Improving accuracy of applications fingerprinting on local networks using NMAP-AMAP-ETTERCAP as a hybrid framework," in Proceedings - 2013 IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2013, 2013, pp. 403–407.

[11] J. Franklin, D. Mccoy, P. Tabriz, V. Neagoe, J. van Randwyk, and D. Sicker, "Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting." Accessed: Jul. 20, 2020.

[12] H. Jafari, O. Omotere, D. Adesina, H. H. Wu, and L. Qian, "IoT Devices Fingerprinting Using Deep Learning," in Proceedings - IEEE Military Communications Conference MILCOM, Jan. 2019, vol. 2019-October, pp. 901–906.

[13] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, and I. Ray, "Behavioral fingerprinting of IoT devices," in Proceedings of the ACM Conference on Computer and Communications Security, Oct. 2018, pp. 41–50.

[14] B. Anderson and D. McGrew, "OS fingerprinting: New techniques and a study of information gain and obfuscation," in 2017 IEEE Conference on Communications and Network Security (CNS), Oct. 2017, vol. 2017-Janua, pp. 1–9.

[15] K. Merchant, S. Revay, G. Stantchev, and B. Nousain, "Deep Learning for RF Device Fingerprinting in Cognitive Communication Networks," IEEE Journal of Selected Topics in Signal Processing, vol. 12, no. 1, pp. 160–167, Feb. 2018, doi: 10.1109/JSTSP.2018.2796446.

[16] Y. Luo, H. Hu, Y. Wen, and D. Tao, "Transforming Device Fingerprinting for Wireless Security via Online Multitask Metric Learning," IEEE Internet of Things Journal, vol. 7, no. 1, pp. 208–219, Jan. 2020, doi: 10.1109/JIOT.2019.2946500.

[17] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy, "DEFT: A Distributed IoT Fingerprinting Technique," IEEE Internet of Things Journal, vol. 6, no. 1, pp. 940–952, Feb. 2019.

[18] X. Wang, L. Gao, S. Mao, and S. Pandey, "CSI-based Fingerprinting for Indoor Localization: A Deep Learning Approach," IEEE Transactions on Vehicular Technology, vol. 66, no. 1, pp. 1–1, Jan. 2016, doi: 10.1109/TVT.2016.2545523.

[19] Q. Xu, R. Zheng, W. Saad, and Z. Han, "Device fingerprinting in wireless networks: Challenges and opportunities," IEEE Communications Surveys and Tutorials, vol. 18, no. 1, pp. 94–104, Jan. 2016.

[20] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Clock around the clock: Time-based device fingerprinting," in Proceedings of the ACM Conference on Computer and Communications Security, Oct. 2018, pp. 1502–1514.

[21] M. Zhang, X. de Carne De Carnavalet, L. Wang, and A. Ragab, "Large-scale empirical study of important features indicative of discovered vulnerabilities to assess application security," IEEE Transactions on Information Forensics and Security, vol. 14, no. 9, pp. 2315–2330, Sep. 2019.

[22] H. Cai, V. W. Zheng, and K. C. C. Chang, "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications," IEEE Transactions on Knowledge and Data Engineering, vol. 30, no. 9, pp. 1616–1637, Sep. 2018.

[23] B. Rozemberczki, O. Kiss, and R. Sarkar, "An API oriented open-source Python framework for unsupervised learning on graphs," 2020.

[24] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning Distributed Representations of Graphs," 2017, doi: 10.1145/1235.

[25] H. Chen and H. Koga, "GL2vec: Graph embedding enriched by line graphs with edge features," in International Conference on Neural Information Processing (ICONIP), Dec. 2019, vol. 11955 LNCS, pp. 3–14.

[26] A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, and E. Müller, "NetLSD: Hearing the shape of a graph," in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Jul. 2018, pp. 2347–2356.

[27] F. Gao, G. Wolf, and M. Hirn, "Geometric scattering for graph data analysis," 2019. Accessed: Apr. 16, 2020.

[28] N. de Lara and E. Pineau, "A Simple Baseline Algorithm for Graph Classification," Oct. 2018, Accessed: Apr. 16, 2020.

[29] S. Verma and Z. L. Zhang, "Hunt for the unique, stable, sparse and fast feature learning on graphs," 2017. Accessed: Apr. 16, 2020.

[30] P. J. Rousseeuw, "Finding Groups in Data: An Introduction to Cluster Analysis (Wiley Series in Probability and Statistics)." Accessed: Jul. 22, 2020.

[31] T. Caliński, J. Harabasz, and T. Caliliski, "A dendrite method for cluster analysis," COMMUNICATIONS IN STATISTICS, vol. 3, no. 1, pp. 1–27, 1974.

[32] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 2, pp. 224–227, 1979.

[33] V. M. Vergara, M. Salman, A. Abrol, F. A. Espinoza, and V. D. Calhoun, "Determining the number of states in dynamic functional connectivity using cluster validity indexes," Journal of Neuroscience Methods, vol. 337, p. 108651, May 2020.

[34] R. Xu, J. Xu, and D. C. Wunsch, "A comparison study of validity indices on swarm-intelligence-based clustering," IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 42, no. 4, pp. 1243–1256, 2012.