Identifying Vulnerable GitHub Repositories and Users in Scientific Cyberinfrastructure: An Unsupervised Graph Embedding Approach

Ben Lazarine
Management Information Systems
University of Arizona
Tucson, AZ, United States
benlazarine@email.arizona.edu

Sagar Samtani
Operations and Decision Technologies
Indiana University
Bloomington, IN, United States
ssamtani@iu.edu

Mark Patton
Management Information Systems
University of Arizona
Tucson, AZ, United States
mpatton@email.arizona.edu

Hongyi Zhu
Information Systems and Cyber Security
University of Texas at San Antonio
San Antonio, TX, United States
hongyi.zhu@utsa.edu

Steven Ullman
Management Information Systems
University of Arizona
Tucson, AZ, United States
stevenullman@email.arizona.edu

Benjamin Ampel
Management Information Systems
University of Arizona
Tucson, AZ, United States
bampel@email.arizona.edu

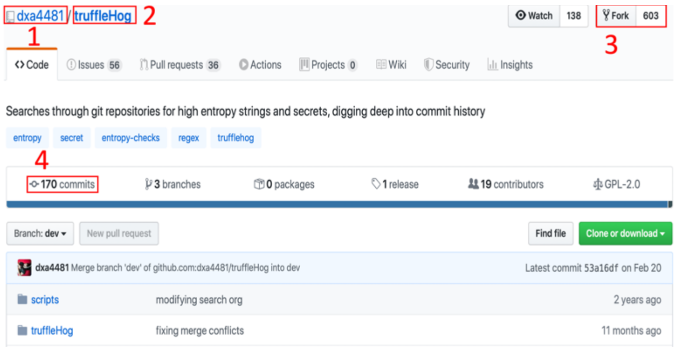
Hsinchun Chen
Management Information Systems
University of Arizona
Tucson, AZ, United States
hchen@eller.arizona.edu

***Abstract*—The scientific cyberinfrastructure community heavily relies on public internet-based systems (e.g., GitHub) to share resources and collaborate. GitHub is one of the most powerful and popular systems for open source collaboration that allows users to share and work on projects in a public space for accelerated development and deployment. Monitoring GitHub for exposed vulnerabilities can save financial cost and prevent misuse and attacks of cyberinfrastructure. Vulnerability scanners that can interface with GitHub directly can be leveraged to conduct such monitoring. This research aims to proactively identify vulnerable communities within scientific cyberinfrastructure. We use social network analysis to construct graphs representing the relationships amongst users and repositories. We leverage prevailing unsupervised graph embedding algorithms to generate graph embeddings that capture the network attributes and nodal features of our repository and user graphs. This enables the clustering of public cyberinfrastructure repositories and users that have similar network attributes and vulnerabilities. Results of this research find that major scientific cyberinfrastructures have vulnerabilities pertaining to secret leakage and insecure coding practices for high-impact genomics research. These results can help organizations address their vulnerable repositories and users in a targeted manner.**

***Keywords*—GitHub; vulnerability scanning; scientific cyberinfrastructure; graph embedding**

I. Introduction

Scientific cyberinfrastructure (CI) has become highly collaborative, with organizations exchanging over 100 gigabytes of data per second and hosting their code bases on social coding repositories (e.g., GitHub, Stack Overflow, etc.) to accelerate collaboration [1]. Well-known NSF-funded projects that host their code bases on GitHub include multiple include the National Center for Atmospheric Research, the Vera C. Rubin Observatory, the National Ecological Observatory Network, and others. GitHub has also been heavily used by a growing number of software developers to share and collaborate on code [2]. It currently has 36 million users that have generated 100 million repositories in 49 million projects [3]. A sample screenshot of a GitHub repository is shown in Figure 1.



Searches through git repositories for high entropy strings and secrets, digging deep into commit history

dxa4481 Merge branch 'dev' of github.com:dxa4481/truffleHog into dev Latest commit 53a16df on Feb 20
scripts modifying search org 2 years ago
truffleHog fixing merge conflicts 11 months ago

Fig. 1. GitHub Repository Example. Each repository has a (1) username, (2) repository name, (3) number of times it has been forked, and (4) commits.

GitHub data can be categorized as relevant to a repository or to a user. Repository metadata includes the owner, forks, and commits. A fork is created when a user "forks" a repository, copying the contents of the repository into a repository in their own account. This enables users to modify the contents of the repository without affecting the original forked repository. A commit indicates a deletion, modification, or addition to a repository made by a user. User metadata can also be queried, including their public repositories, e-mail, name, organization, and account age. With social coding repositories, the pace of software development in CI continues to accelerate. However, this rapid growth has also led to CI users inadvertently engaging in insecure coding practices, accidentally posting confidential secrets (e.g., passwords), and exposing numerous other vulnerabilities. Each of these issues can enable a malicious hacker to execute cyber-attacks that result in significant and often irreversible scientific CI information and financial losses.

In light of these potentially significant ramifications, we propose a novel research framework to identify vulnerable groups of repositories and users within a major scientific CI community. This framework draws upon state-of-the-art techniques in vulnerability assessment, social network analysis, graph embedding, and clustering. Our research provides a principled approach for CI security administrators to proactively identify groups of repositories and users with related types of vulnerabilities for targeted remediation and mitigation.

The remainder of this paper is organized as follows. First, we review literature on social coding repository, social network analysis, graph embedding techniques, and vulnerability scanning approaches. Second, we present our research design. Third, we summarize our experimental results. Finally, we discuss future directions and conclude this research.

II. Literature Review

We review three key areas of literature to guide this research. First, we review social coding repository research to identify past efforts on detecting vulnerabilities in social coding platforms. Second, we review social network analysis, focusing on bipartite networks, and graph embedding research. Finally, we review vulnerability scanning methods to identify past methods used to scan social coding platforms for vulnerabilities.

A. Social Coding Repository Research

Past social coding research has focused primarily on high-level user analysis and low-level code snippet analysis. Most work has utilized GitHub and Stack Overflow data to conduct cross platform analysis, detect insecure code snippets, examine secret leakage, and identify key users [2], [4-8]. Past research that has used GitHub focused on users or repositories, but not both. Limited research utilizes social network analysis and vulnerability scanning concurrently to identify key groups of repositories and/or possessing similar vulnerabilities. Past research that focused on user identification did not conduct any vulnerability assessment. The relationship between repositories and users on GitHub is conducive to a social network analysis approach to identify influential repositories and users to enhance the results of vulnerability scans.

B. Social Network Analysis and Graph Embedding Techniques

Social network analysis is used to study relationships between connected entities. A social network is comprised of nodes and edges. Nodes are entities and edges are their relationships. Nodes can be labeled with features. If a network contains two clearly defined node types (e.g., repositories and users), a bipartite network configuration can be used to present it. A bipartite network is comprised of two sets of nodes having no edges connecting any two nodes in the same set. The value of a bipartite network representation is that it can project into two monopartite networks. Each projected network shows relationships between nodes of the same type. For example, if two repositories, A and B, both have a connection to the same user, C, then the repository monopartite graph projection will have an edge connecting A and B. Generating monopartite graphs enables calculation of key nodal and topological level attributes (e.g., degree, betweenness, graph density, etc.).

Constructing a network allows for clear identification of key users and repositories. However, identifying groups of vulnerable users and repositories requires embedding nodes based on their relationships and the vulnerabilities they possess. Graph embedding techniques utilize these network attributes to generate graph embeddings that represent the network in a low-dimensional space [9]. Graph embedding methods typically operate in a supervised or unsupervised fashion. The selection of graph embedding methods is contingent upon the graph formulation (e.g., directed or undirected), available data (e.g., nodal features), and whether a priori knowledge (i.e., labels) is available. Since our proposed application has limited label data, grouping repositories and users based on their vulnerabilities requires an unsupervised approach.

Unsupervised graph embedding methods typically operate on one of four functions to create embeddings: matrix factorization (e.g., DeepWalk) [9], random walk (e.g., Node2vec) [10], deep representation learning (e.g., Graph Convolutional Autoencoder (GCAE)) [11], or edge reconstruction (e.g., Large-scale Information Network Embedding (LINE)) [12]. Matrix factorization is preferable for the proposed research as it can account for graph and nodal features when generating its embeddings. This supports analysis of vulnerabilities associated with the repositories and users in our graph. Other approaches do not provide this capability. Additionally, embeddings generated by matrix factorization are suitable for downstream tasks including classification and clustering. As a result, they can facilitate the proposed analysis of identifying groups of vulnerable users and repositories.

C. Vulnerability Scanning

Traditional vulnerability scanning for IT environments consist of scanners (e.g., Nessus, Burp Suite) that monitor network traffic and examine IoT device characteristics [13-14]. Vulnerability scanning also includes scanning software for vulnerabilities [15]. Software vulnerability scanning research focuses on two major areas: scanning software as it is being developed and scanning published (static) software.

Social coding repositories consist of published software. Static vulnerability scanners have been developed to scan popular coding languages (e.g., Python, C, etc.) for insecure coding practices [16-17] as well as scanning multiple file types for secrets [4]. Scanning tools are selected based on the programming language, coverage, and goal of the vulnerability scan. However, there is limited research that scans social coding repositories for insecure coding practices using static vulnerability scanners. Moreover, how these vulnerabilities can be included as features in networks representing repositories and users to create fine-grained groups has not yet been studied.

D. Research Gaps and Questions

We identified two key research gaps from our literature

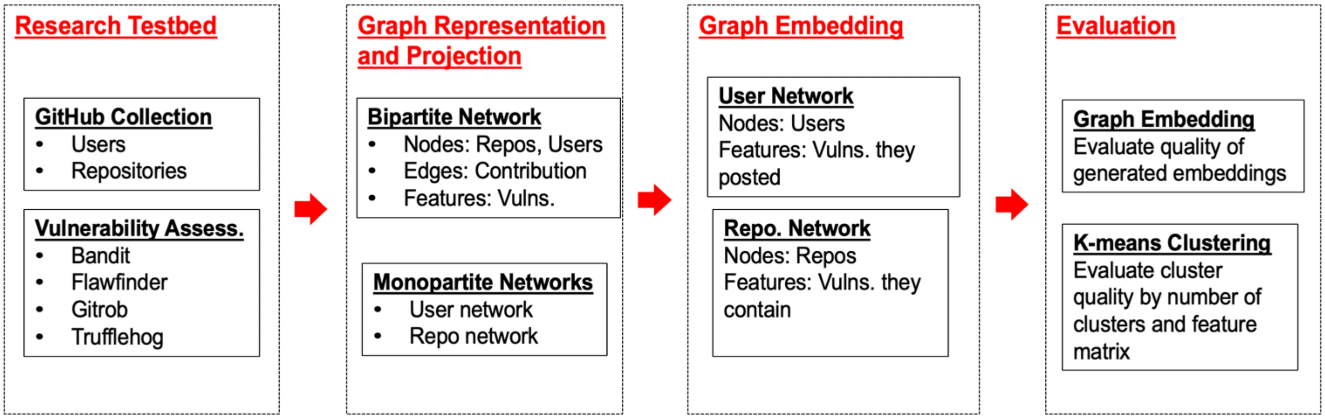


Fig. 2. Proposed Research Framework.

review. First, research identifying insecure code present in social coding platforms has been conducted. However, we found no large-scale vulnerability assessment study of a large scientific cyberinfrastructure user community. Second, limited research has combined social network analysis and vulnerability assessment to identify groups of users and repositories based on their relationships and vulnerabilities. Based on these research gaps we propose the following questions:

- How can we identify vulnerabilities within GitHub repositories in scientific cyberinfrastructure?
- How can we pinpoint key repositories and users based on their vulnerabilities and relationships?
- How can we group repositories and users based on their vulnerabilities and relationships in scientific cyberinfrastructure?

III. Research Design

We present our proposed research framework in Figure 2. The framework has four major components: Research Testbed, Graph Representation and Projection, Graph Embedding, and Evaluation. We describe each in the following sub-sections.

A. Research Testbed: GitHub Collection and Vulnerability Assessment

We identified a large-scale and long-standing scientific CI for collection and analysis. We anonymized their name to protect their privacy. This NSF-funded CI provides computational resources to an international user-base of over 75,000 life scientist researchers working with large datasets and conducting high-impact scientific analysis (e.g., genomics, black hole imaging). The CI has 258 repositories on GitHub relevant to their codebase, environment, and documentation that are used by 2,549 users across 424 organizations.

Our research testbed for the targeted scientific CI comprises of GitHub collection and vulnerability assessment. For the former, we used GitHub's API to collect all GitHub data relevant to the community, including repositories and users and their full history of development. Our GitHub collection consists of four groups of repositories: repositories created by the main GitHub account (root repositories), forks of root repositories, repositories returned by querying GitHub's search API with the name of the organization as a keyword (searched repositories), and forks of searched repositories. We also identified 2,401 accounts interacting with the host organization's GitHub ecosystem by making a change (commit) to any repository in our collection. A breakdown of the four repository types is shown in Table I.

TABLE I. Summary of Selected GitHub Repositories

	Root Repositories	Forks of Root	Searched Repositories	Forks of Searched	Total
Repositories	84	244	174	121	623
Forks	244	8	121	0	373
Contributors	226	93	152	92	563
Issues	132	39	44	37	252
Languages	45	25	42	25	54 (distinct)
Top 3 Languages	C++, Python, C	Python, JavaScript, Shell	Python, HTML, C	Python, Java, Clojure	Python, JavaScript, Java

Python and C/C++ are the most common languages used in this CI (accounting for 56% of all code). Therefore, we identify software vulnerability scanners developed for those languages. We reviewed 14 vulnerability scanners and selected four: Bandit, Flawfinder, Gitrob, and Trufflehog. These were selected based on their coverage, usability, age, and GitHub usage [4][16-17]. Bandit scans Python files with 70 vulnerability tests, Flawfinder scans C/C++ files for 18 different common weakness enumerations (CWEs) outlined by MITRE, Gitrob scans GitHub for secrets and log files, and Trufflehog scans GitHub for secrets. Gitrob and Trufflehog are designed specifically to for GitHub. Therefore, they can scan repository histories and link detected vulnerabilities to the users who posted them.

The four identified scanners return 13 major types of vulnerabilities. These can be broadly categorized into three groups: secrets, insecure coding, and attack susceptibilities. Secret vulnerabilities include potential password/key leakage, an occurrence of the string "password" in a file, weak cryptography, and filetypes known to contain secrets (i.e. configuration files). Insecure coding vulnerabilities include bad file permissions, insecure functions, insecure modules, deprecated libraries, and insecure internet connections. Finally, attack susceptibilities include insecure user inputs, SQL injections, XML attacks, and XSS attacks.

Overall, Bandit returned 31,375 vulnerabilities across 188 repositories. Key vulnerabilities returned by Bandit included 25,678 insecure functions, 1,579 insecure modules, 1,352 secrets, and 1,089 insecure inputs. Flawfinder identified 27,200 vulnerabilities across 71 repositories. These included 23,935 insecure inputs, 2,055 insecure functions, 972 insecure permissions, and 170 weak cryptographic instances. Gitrob and identified 121 secret instances and 80 configuration/log files across 13 repositories. Trufflehog identified 7,381 secret instances across 623 repositories.

B. *Graph Representation and Projection*

The relationship between users and repositories is suitable for bipartite network analysis. The value of representing this relationship in a bipartite network is that it can be projected into two monopartite networks to identify key users and repositories. We formally denote our bipartite graph as G=(U, R, E, F) where G is a directed graph, *U* is the node set, $\{u_1, u_2, u_3, \ldots u_n\}$, of all users that have contributed to a repository, *R* is the node set, $\{r_1, r_2, r_3, \ldots r_n\}$, of all repositories, *E* is the edge set, $\{e_1, e_2, e_3, \ldots e_n\}$, of directed edges from a user contributing to a repository, and *F* is the feature matrix of each node representing the number of vulnerabilities each user or repository possesses.

We defined two feature sets: one for users and one for repositories. The user feature set includes potential passwords/keys, occurrences of the string "password" in a file, and sensitive filetypes. Labeling users with these features supports clustering analysis of users with similar secret leakage behavior. The repository feature set included all 13 identified vulnerabilities. The defined repository feature set supports clustering analysis of repositories with similar vulnerabilities across all three types of vulnerabilities identified in our scan. We projected the bipartite network into repository and user monopartite graphs. Monopartite graphs support analysis of key topological and node level metrics. We summarize each metric and their corresponding security implications in Table II.

TABLE II. SUMMARY OF SELECTED TOPOLOGICAL AND NODE LEVEL METRICS AND THEIR SECURITY IMPLICATIONS

Category	Metric	Definition	Security Implications
Network	Graph Density	Sum of edges divided by total possible edges	Inter-dependence of repositories/users
	Diameter	Maximum geodesic distance from a node to all other nodes	Indicates the breadth and diversity of a repository/user
	Average path length	Average distance between two nodes	Average dependencies between repositories/users
Node	Number	# of network nodes	Number of repositories/users
	Overall Degree	Sum of a node's in and out degree	Overall importance of the repository/user
	Betweenness	Proportion of shortest paths passing through a node.	Repository that has many shared contributors/User that commits to many repositories

C. *Graph Embedding and Evaluation*

Grouping users and repositories based on their relationships and vulnerabilities without a priori knowledge requires an unsupervised graph embedding method that accounts for textual nodal features and operates on undirected graphs. Therefore, we select text associated Deep Walk (TADW) [9] to generate graph embeddings for the repository and user graphs. The TADW embedding process is as follows:

- **Step 1:** Matrix factorization is used to learn vertex representation according to network structure.
- **Step 2:** Node feature matrix is obtained from the dataset.
- **Step 3:** The vertex matrix and feature matrix are concatenated to build a unified 2k-dimensional vertex embedding matrix for network representation.
- **Step 4:** Steps 1-3 repeat for all graphs in a set.

Evaluating embedding quality is essential for ensuring that the downstream task (in this study, clustering) performs well. Therefore, we evaluate the quality of the graph embeddings with Mean Average Precision (MAP). MAP evaluates how well a graph embedding model reconstructs the original graph by calculating the average precision of each node [18]. MAP returns a scalar value from zero to one. Higher quality embeddings have a MAP close to one. Scores 0.70 and higher are commonly accepted as high-quality. Second, we evaluate the quality of graph embedding clusters generated with k-means clustering. Cluster quality is evaluated based on three well-known clustering measures: silhouette, Calinski-Harabasz (CH), and Davies Bouldin (DB) [19]. Silhouette represents the possible data clusters using average dissimilarity [20]. CH examines inter-cluster separation and intra-cluster compactness as a ratio [21]. DB calculates the ratio of intra-cluster to inter-cluster distances [22]. High quality clusters have a silhouette close to one, a high CH, and a DB close to zero.

IV. RESULTS AND DISCUSSION

We present our results below. First, we discuss our GitHub graph representation and projection findings. Second, we discuss our graph embedding evaluation results for both our user and repository networks. Finally, we present our clustering images and interpret selected key clusters.

A. *Graph Representation and Projection*

Topological and node level summary statistics for the overall bipartite graph and each monopartite graph projection are presented in Table III.

TABLE III. SUMMARY STATISTICS FOR BIPARTITE GRAPH AND MONOPARTITE GRAPH PROJECTIONS

Category	Metric	Bipartite (Repository and User) Graph	Monopartite Repository Graph	Monopartite User Graph
Network	Number of Nodes	3,019	618	2,401
	Number of Edges	8,606	35,703	1,065,599
	Graph Density	0.002	0.19	0.37
	Network Diameter	13	6	6
	Average Path Length	4.43	2.07	1.81
Node	Max Degree	1,342	294	1,942
	Min Degree	0	0	0
	Average Degree	5.7	115.54	887.63
	Average Betweenness	4,511.94	203.63	52.41

We make several key observations from these results. First, both repository and user networks have relatively high average degrees of 115 and 887, respectively. This indicates that the majority of nodes in both graphs have importance to the network. Second, the user graph has a higher density than the repository graph. This suggests that users have varying interests across all repositories, but within groups of repositories, users have a high interdependence. Finally, we observe that the user graph has a low average path length, meaning that all users can be connected via a short path (1.81). This indicates an active user base across our CI's repositories.

B. *Graph Embedding Evaluation*

MAP was calculated for TADW-generated repository and user graph embeddings. Each graph's full feature set was used when generating the embeddings. The user and repository embeddings achieved strong MAP scores of 0.94 and 0.74, respectively. As such, we cluster the TADW-generated embeddings and evaluate for both repositories and users.

C. *Clustering Results*

We evaluated k-means clustering for cluster sizes four to nine and compared the quality of the clusters in terms of silhouette, CH, and DB. As cluster size increases, the quality of the clusters improves; silhouette increases, CH remains high and DB decreases. Performance peaked at k=9 for both repositories and users with a silhouette of 0.27 and 0.79, CH of 66 and 1,479, and DB of 1.58 and 1.28, respectively. Therefore, we use k=9 to generate clustering results for both repositories and users. Clustering results are presented in Figure 3. Each cluster is color-coded, circled, and labeled.

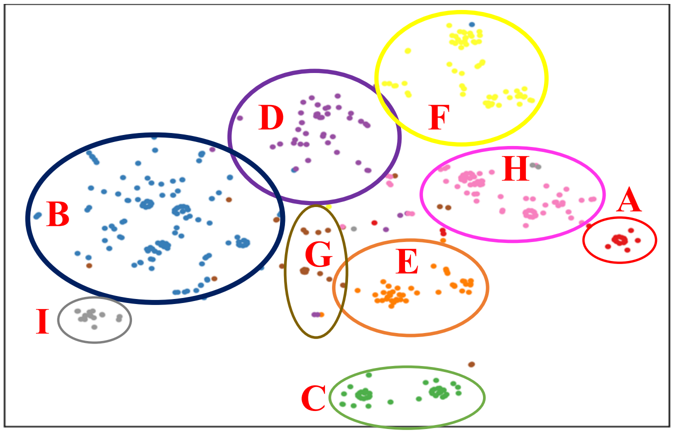


Fig. 3. Repository Clusters.

Overall, the maximum repository cluster size was 177, average was 76, and minimum was 20. Figure 3 illustrates that the repository clusters are well separated and have relatively small intra-cluster distances. Cluster A is 87% searched repositories. Cluster B is 83% root forks. Clusters C, E, and I are primarily fork repositories (91, 68, and 69%, respectively). All repository types are represented in clusters D and G. Cluster F is 79% root and searched repositories. Cluster H is 97% searched fork repositories. We summarize the number of each vulnerability in each cluster in Figure 4 to further understand their composition.

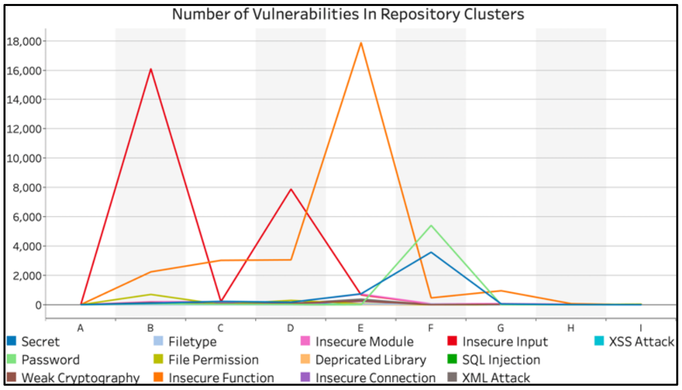


Fig. 4. Number of Vulnerabilities in Repository Clusters.

Clusters A, G, H and I have less than 1,000 vulnerabilities. Clusters B and D have a high number of insecure input vulnerabilities with over 7,000 occurrences each. Clusters C and E have a high number of insecure function vulnerabilities with 3,031 and 17,898, respectively. Lastly, secret related vulnerabilities are primarily present in cluster F, with 5,415 instances of the string "password" and 3,586 potential secrets/keys. Since the prevailing vulnerabilities are insecure functions, insecure input, and secret leakage, we examine these closer to examine the potential threat they pose to CI.

High severity insecure function vulnerabilities include the use of '*mktemp'* in Python to create temporary files and '*printf*' in C which accepts a format string from an external source. High severity insecure input vulnerabilities include '*yaml load*' in Python and '*strcopy*' in C. '*Mktemp'* allows an attacker to modify a file before it is opened. Similarly, '*yaml load'* can allow remote code execution to cause research data loss. Repositories with this vulnerability are commonly used for genome sequencing. Breaching such research can result in years of research assets being irrevocably lost. High severity secret vulnerabilities include SSH and API key leakages. '*Printf*' and '*strcopy'* can lead to buffer overflow and data representation issues. Both could result in CI downtime, hindering research speed. High severity secret vulnerabilities include SSH and API key leakages. SSH keys are used to access a private machine with the full administrator permissions. Leaked API keys (e.g., AWS keys) enable hackers' access to APIs with CI resources. A hacker successfully accessing a CI VM or having unauthorized API usage could lead to significant abuse of the resources. A common example is running heavy Bitcoin mining computations. These processes are a common concern for the CI community due to their significant financial ramifications [1].

Due to the five identified vulnerable clusters having different types of repositories, different mitigation strategies can be taken for each cluster. Insecure function and insecure input vulnerabilities primarily being in fork repositories suggests that when users generate forks, they preserve the vulnerabilities present in the repository at that time. Thus, while the vulnerabilities in the root repository have been addressed, they still exist across the forks. These vulnerabilities across clusters B, C, and E can be addressed by submitting downstream pull requests to the fork repositories on GitHub. This issues a request to forks to pull updates that have been made to the root repository. For root repositories in cluster F that have secret vulnerabilities, secrets committed to GitHub remain in repository history. Therefore, addressing secret leakage requires identifying and replacing compromised secrets.

Selected user clustering results are presented in Figure 5. There are 2,401 users in nine clusters with a maximum size of 1,413 users , average of 262, and minimum of 15. The clusters are color-coded, circled, and labeled.

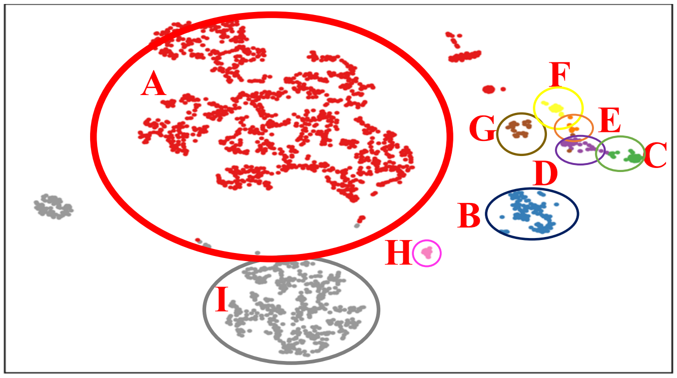


Fig. 5. Illustration of User Clustering Results (k=9).

Due to the nature of our vulnerability scan, only secret leakage could be directly linked to users. Clusters A, E, F, G, H and I all have less than 25 secrets present. Clusters B, C, and D are comprised of 226 users linked to 2,470 SSH private key instances. Additionally, cluster D contains 72 API key instances and 85 configuration files. These vulnerabilities pose the same threat to CI as the repository secret vulnerabilities. However, because they are linked to users, the mitigation strategy can be enhanced. In addition to identifying and replacing these secrets, CI communities can provide targeted security awareness trainings to educate their user groups that are linked to secret leakages. This can prevent users from posting secrets to GitHub.

V. Conclusion and Future Directions

In this research, we present a novel graph embedding framework to automatically identify groups of vulnerable CI repositories and users for subsequent targeted mitigation. We showed that CI organizations present vulnerabilities in their collaboration spaces and vulnerable communities can be detected. Future research can analyze multiple organizations of different sizes and scopes. Developing an interactive portal for organizations to automatically explore their vulnerabilities is also a promising research direction. Each direction can help facilitate enhanced scientific CI cybersecurity.

Acknowledgment

This material is based upon work supported by the NSF under Grant Numbers DGE-1921485 (SFS), OAC-1917117 (CICI), and CNS-1850362 (CRII and SaTC).

References

[1] V. Welch, S. Sons, J. Marsteller, R. Biever, I. Kouper, and M. Corn, "Trusted CI webinar: Securing Scientific Cyberinfrastructure: The ResearchSOC," 2019.

[2] Y. Fan, Y. Zhang , S. Hou, L. Chen, Y. Ye, C. Shi, L. Zhao, and S. Xu, "idev: Enhancing social coding security by cross-platform user identification between GitHub and stack overflow," in IJCAI 2019, pages 2272–2278, 2019.

[3] GitHub.com

[4] M. Meli, M.R. McNiece, and B. Reaves, "How bad can it git? Characterizing secret leakage in public GitHub repositories," in 26th Annual Network and Distributed System Security Symposium, NDSS 2019.

[5] Y. Ye, S. Hou, L. Chen, X. Li, L. Zhao, S. Xu, and Q. Xiong, "ICSD: An Automatic System for Insecure Code Snippet Detection in Stack Overflow over Heterogeneous Information Network," in ACSAC, pages 542–552. ACM, 2018.

[6] R. Bana and A. Arora, "Influence Indexing of Developers, Repositories, Technologies and Programming languages on Social Coding Community GitHub," in Eleventh International Conference on Contemporary Computing (IC3), pages 1–6, 2018.

[7] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. A. Argoty, "Secure coding practices in Java: Challenges and vulnerabilities," in ICSE, 2018.

[8] D. Yang, P. Martins, V. Saini, and C.V. Lopes, "Stack Overflow in GitHub: Any Snippets There?" in 14th International Conference on Mining Software Repositories (MSR 2017), pages 280–290, 2017.

[9] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network Representation Learning with Rich Text Information" In Proceedings of the 24th International Joint Conference on Artificial Intelligence, 2015.

[10] A. Grover and J. Leskovec "node2vec: Scalable Feature Learning for Networks." In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining – KDD '16, pages 855–864, 2016.

[11] T. Kipf and M. Welling "Semi-Supervised Classification with Graph Convolutional Networks." The International Conference on Learning Representations, 2017.

[12] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale Information Network Embedding." In Proceedings of the 24th International Conference on World Wide Web – WWW '15, pages 1067–1077, 2015.

[13] S. Samtani, S. Yu, H. Zhu, M. Patton, and H. Chen, "Identifying SCADA vulnerabilities using passive and active vulnerability assessment techniques," in 2016 IEEE International Conference on Intelligence and Security Informatics (ISI), pages 25-30, 2016.

[14] E. McMahon, M. Patton, H. Chen, and S. Samtani, "Benchmarking Vulnerability Assessment Tools for Enhanced Cyber-Physical System (CPS) Resiliency," in 2018 IEEE International Conference on Intelligence and Security Informatics (ISI),pages 100-105, 2018.

[15] L. Neil, S. Mittal, and A. Joshi, "Mining Threat Intelligence about Open-Source Projects and Libraries from Code Repository Issues and Bug Reports," in 2018 IEEE International Conference on Intelligence and Security Informatics (ISI),pages 7-12, 2018.

[16] A. Kaur and R. Nayyar, "A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code." 3rd International Conference on Computing and Network Communications, pages 2023-2029, 2020.

[17] K. A. Torkura and C. Meinel, "Towards Vulnerability Assessment as a Service in OpenStack Clouds," IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops), Dubai, pages 1-8, 2016.

[18] P. Ciu, X. Wang, J. Pei, and W. Zhu, "A Survey on Network Embedding," IEEE TKDE, 2018.

[19] 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, page 108651, May 2020.

[20] P. J. Rousseeuw, "Finding Groups in Data: An Introduction to Cluster Analysis (Wiley Series in Probability and Statistics)." 1990.

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

[22] 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.