User Profiling and Vulnerability Introduction Prediction in Social Coding Repositories: A Dynamic Graph Embedding Approach

Agrim Sachdeva, Ben Lazarine, Hongyi Zhu, Sagar Samtani
Indiana University, University of Texas at San Antonio

Monday, August 7, 2023

*This material is based upon work supported by the National Science Foundation under Grant No. NSF OAC-1917117 (CICI)*
Introduction: Software Vulnerabilities Due to Human Error

• Introduction of several vulnerabilities is caused by human error
  • Most software vulnerabilities are mistakes, not malicious attacks
  • Critical-severity vulnerability make it through code review just as easily as low-severity ones
  • Vulnerabilities typically go undetected for 218 weeks (over four years) before being disclosed (Fig. 1)

• Context: Open Source and GitHub
  • Open-source code is vital to the global economy; services and technology from banking to healthcare; > $100 billion impact (Fig. 1)
  • Many scientific CIs and their users host and share their research source code on Social Coding Repositories (SCR) such as GitHub
  • Susceptible due to distributed development; evolving risks; zero-day vulnerabilities

• “Shifting left” is an important mitigation strategy
  • Early detection, or the prediction of the introduction of vulnerabilities by users

Figure 1. Lifecycle of a vulnerability

Sources: 
1: https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html
2: https://blog.gitguardian.com/state-of-secrets-sprawl-2021/
3. State of the Octoverse, GitHub, 2020
Current Solutions

Code-first
• DevSecOps-enabled: Integrated vulnerability scanning or detection systems
• Dependabot, Dependency Review, Dependency Graph, CodeQL and Secret Scanning
Identify vulnerabilities after they have been committed to code
Work at the repository-level, and do not focus on users or provide feedback
Require extensive configuration

User-first
• Developer security awareness training

More effective in reducing vulnerabilities than embedded tools in interfaces (Sedova, 2017)
Quickly loses efficacy if non-targeted
Causes training fatigue

Shift from detecting vulnerabilities in code after they have been introduced to predicting user errors and prevent introduction of vulnerabilities
Example GitHub User

Repositories can be "starred"

Fig. 2. A GitHub Profile
Vulnerabilities can be introduced and propagated by users.

Vulnerability introduction can be impacted by:
• Direct and indirect exposure
• Propagation of knowledge and information between developers

Objective: Predict the introduction of a vulnerability by a user into a repository.

Can enable proactive risk management, e.g., targeted security awareness trainings

Fig. 3. Example Vulnerability Introduction Prediction within the organization Cyverse
Literature Review: Vulnerability Management for SCR

<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Platform/ Dataset</th>
<th>Focus</th>
<th>Method</th>
<th>User Profiling</th>
<th>Vulnerability Assessment</th>
<th>Temporal Dynamics</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022</td>
<td>Shahid et al.</td>
<td>Network Metadata</td>
<td>Hybrid CNN + Cookie Analysis</td>
<td>CNN</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2022</td>
<td>Zerouali et al.</td>
<td>GitHub: npm, RubyGems</td>
<td>Vulnerability dependency networks</td>
<td>Empirical</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2021</td>
<td>Shaji et al.</td>
<td>GitHub: organizations</td>
<td>Non-intrusive vulnerability detection</td>
<td>Probabilistic model</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2021</td>
<td>Rabheru et al.</td>
<td>GitHub: Wordpress</td>
<td>Novel vulnerability detection</td>
<td>GRU + GCN</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2021</td>
<td>Mazeura-Rozo et al.</td>
<td>Source Code Representations</td>
<td>Comparing DL to ML models, e.g., Google’s AutoML</td>
<td>Empirical</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>2020</td>
<td>Lazarine et al.</td>
<td>GitHub</td>
<td>Cluster attributes and vulnerabilities</td>
<td>TADW</td>
<td>Yes</td>
<td>4 scanners</td>
<td>No</td>
</tr>
<tr>
<td>2020</td>
<td>Zhang et al.</td>
<td>GitHub</td>
<td>Malicious blockchain repository detection</td>
<td>HIN</td>
<td>No</td>
<td>3 scanners</td>
<td>No</td>
</tr>
<tr>
<td>2019</td>
<td>Gong et al.</td>
<td>GitHub</td>
<td>Detection of malicious online accounts</td>
<td>Phased LSTM</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2019</td>
<td>Meli et al.</td>
<td>GitHub</td>
<td>Data leakage</td>
<td>Regular Expressions</td>
<td>No</td>
<td>1 scanner</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1. Literature Review of studies that use GitHub data for vulnerability analysis or modeling users


- **Key Observations:**
  1. Most recent studies do not focus on prediction of vulnerabilities, or incorporate a vulnerability assessment on code, overlooking the incidence and nature of vulnerabilities.
  2. Although GitHub as a time-evolving interaction network, studies do not capture the temporal dynamics need to be captured and modelled, reducing the accuracy of the data representation, and therefore, the underlying phenomenon.
  3. The results of vulnerability analysis contains valuable information such as severity levels, which most studies omit.
GitHub as a time-evolving interaction network wherein individual edges and nodes are inserted or deleted over time in a continuous manner.

Based on the characteristics of our data, i.e., a time-stamped dynamic interaction network, we review dynamic graph representation learning techniques.

By representing users and repositories as nodes, and vulnerabilities as links between the nodes, the task of link prediction would help us predict the introduction of vulnerabilities.
User embeddings: vulnerability propagation score/ propensity to introduce vulnerabilities
Repository embeddings: Risk level, or vulnerability level

The propagation of information is modeled as non-linear node dynamic evolution between interactions, which captures the knowledge and communication between users that impacts the introduction of a vulnerability.
## Literature Review: Dynamic Graph Representation Learning

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Methods</th>
<th>Algorithm</th>
<th>Contribution</th>
<th>Discrete/Continuous</th>
<th>Information Propagation</th>
<th>Concurrent Interactions</th>
<th>Authors</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evenly-Spaced Snapshot Sequence</td>
<td>Dynamic Latent Space Models</td>
<td>DSNL</td>
<td>Forward-backward algorithm on the Markov chain over timesteps</td>
<td>Discrete</td>
<td>No</td>
<td>No</td>
<td>Sarkar and Moore</td>
<td>2005</td>
</tr>
<tr>
<td></td>
<td>Incremental SVD</td>
<td>TIMERS</td>
<td>Set the restart time to reduce accumulated error</td>
<td>Discrete</td>
<td>No</td>
<td>No</td>
<td>Zhang et al.</td>
<td>2018</td>
</tr>
<tr>
<td>Unevenly-Spaced Snapshot Sequence</td>
<td>Random Walk Based Methods</td>
<td>CTDNE</td>
<td>Temporal random walks that contain a sequence of edges in order</td>
<td>Continuous</td>
<td>No</td>
<td>No</td>
<td>Nguyen et al.</td>
<td>2018</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DNE</td>
<td>Extension for the Skip-gram based network embedding methods</td>
<td>Discrete</td>
<td>No</td>
<td>No</td>
<td>Du et al.</td>
<td>2018</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DynamicTriad</td>
<td>Learn dynamic embeddings by modeling the triadic closure</td>
<td>Discrete</td>
<td>No</td>
<td>No</td>
<td>Zhou et al.</td>
<td>2018</td>
</tr>
<tr>
<td></td>
<td>Graph Neural Network Methods</td>
<td>DyRep</td>
<td>Temporal attention layer to capture the neighbors’ interactions</td>
<td>Continuous</td>
<td>No</td>
<td>No</td>
<td>Trivedi et al.</td>
<td>2019</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JODIE</td>
<td>Coupled recurrent neural network model; learns embedding trajectories of two types of nodes (e.g., users and items)</td>
<td>Continuous</td>
<td>No</td>
<td>No</td>
<td>Kumar et al.</td>
<td>2019</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CoPE</td>
<td>Modeling continuous propagation and evolution</td>
<td>Continuous</td>
<td>Yes</td>
<td>Yes</td>
<td>Zhang et al.</td>
<td>2021</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TGAT</td>
<td>Attention layer to efficiently aggregate temporal-topological neighborhood features</td>
<td>Continuous</td>
<td>No</td>
<td>Yes</td>
<td>Xu et al.</td>
<td>2020</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TGN</td>
<td>“Memory” module to update node embeddings</td>
<td>Continuous</td>
<td>No</td>
<td>Yes</td>
<td>Rossi et al.</td>
<td>2020</td>
</tr>
</tbody>
</table>

Table 2. Dynamic Graph Representation Learning Methods; Note: DSNL: Dynamic Social Network In Latent Space; TIMERS: Theoretically Instructed Maximum-Error-bounded Restart of SVD; CTDNE: Continuous-Time Dynamic Network Embeddings; GNN: Graph Neural Networks (GNNs); DyRep: Dynamic Representation; CoPE: Continuous Propagation and Evolution

### Key Observations:

1. Prevailing dynamic graph representation Learning for continuous-time interaction graphs are neural-network based methods, which can further be classified into RNN-based (DyREP, JODIE, CoPE), or Attention-based (TGAT, TGN)

2. CoPE can capture the evolution of nodal embeddings based on the propagation of information, i.e., the impacts of the users on neighboring users and repositories between interactions
Research Questions

Extant security education, training, and awareness (SETA) research does not comment on open-source software security awareness trainings, or the timing of the training to be delivered. The personalization and the timing of delivery are important to security training outcomes.

Baseline link prediction using dynamic graph methods does not consider the severity of the vulnerabilities when they spread, or the relative influence of users.

Based on these research gaps, we pose the following research questions:

1. How can we predict the introduction of vulnerabilities by users into repositories while accounting for information propagation?

2. How can we adapt dynamic graph link prediction methods to incorporate rich feature sets for users, repositories, and vulnerabilities, and capture the relative influence of high-risk repositories and actors?
Research Question

Shortcomings in existing approaches necessitate the need for **personalized and targeted training**

The prediction of vulnerabilities in source code will allow the creation of risk profiles, and enable proactive and personalized security awareness training.

**How can we extend and adapt CoPE to incorporate rich feature sets for users, repositories, and vulnerabilities, and capture the relative influence of high-risk repositories and actors?**
Research Design

1. Data Collection
   - Repository Collection
   - Vulnerability Scanning

   NCAR
   176 users, 77 repos, 3326 vulnerabilities

2. Graph Formulation
   - Feature Extraction
     - User, Repository, Interaction
   - G = (U, R, E)
     - E: Vulnerability Introduction
     - Features: \( f(u_k, r_k, t_k) \)

3. Proposed SeCoPE Model
   - Weighted Continuous Propagation Unit
   - Nodal Embedding Generation
   - Link Prediction

4. Evaluation
   - Experiment 1: seCoPE vs. RNN-based Methods
   - Experiment 2: seCoPE vs. Attention-based Methods

Figure 6. Research Design and Testbed
**Research Design: Data Collection**

**NCAR:** Federally funded R&D center for climate science, atmospheric chemistry, solar-terrestrial interactions; founded in 1956; collaborates with 115 universities; **176 users, 77 repositories, 3326 vulnerabilities**

We selected four open-source vulnerability assessment scanners based on language categories of vulnerabilities they scan for, languages, usability and age, i.e., Bandit, Flaw Finder, Gitrob, Trufflehog

<table>
<thead>
<tr>
<th>Type</th>
<th>Sub-category</th>
<th>Vuln.</th>
<th>Description</th>
<th>Example</th>
<th>Bandit</th>
<th>Flaw Finder</th>
<th>Gitrob</th>
<th>Trufflehog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret</td>
<td>Secret</td>
<td>A potential password/key</td>
<td>73b6afec22ff801a132fc89200a0614953211cd</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Cryptography</td>
<td>Password</td>
<td>Word password found</td>
<td>irods://user:pass@host:port/destination</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Filetype</td>
<td>Filetype</td>
<td>File that may contain secrets</td>
<td>Django configuration file</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Permission</td>
<td>File permission</td>
<td>File may have dang. Permissions</td>
<td>int err_code = chmod(filePath, 0664);</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Insecure</td>
<td>Insecure function</td>
<td>Function can be vulnerable</td>
<td>Use of insecure and deprecated function (mktemp).</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Insecure</td>
<td>Insecure module</td>
<td>Module can be vulnerable</td>
<td>Pickle can be unsafe when used to deserialized untrusted data</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Insecure</td>
<td>Depricated library</td>
<td>Library no longer supported</td>
<td>The pyCrypto library and its module atfork are no longer actively maintained</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Internet</td>
<td>Insecure conn.</td>
<td>Dangerous internet connections</td>
<td>Requests call with verify=False disabling SSL certificate checks, security issue.</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Injection</td>
<td>Insecure input</td>
<td>Dangerous handling of user input</td>
<td>Use of unsafe yaml load. Allows instantiation of arbitrary objects.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Injection</td>
<td>SQL injection</td>
<td>Hardcoded SQL expressions</td>
<td>Possible SQL injection vector through string-based query construction.</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Attack</td>
<td>XML attack</td>
<td>Dangerous XML library</td>
<td>Using xmlrpclib to parse untrusted XML data is known to be vulnerable.</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Attack</td>
<td>XSS vulnerability</td>
<td>Dangerous library usage</td>
<td>By default, jinja2 sets autoescape to False.</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3. Key Vulnerabilities Returned from Scanners
Research Design: Graph Formulation

• Interaction Graph
  ▪ An interaction graph $G = (U \cup R, E)$ is a user-repository bipartite graph, where each edge $e = (u, r, t) \in E$ represents the introduction of a vulnerability

• Interactions
  ▪ Given the user set $U$ and repository set $R$, sequential interactions between users and repositories can be organized as ordered set $E = \{(u_k, r_k, t_k)\}_{k=1}^n$, where $u_k \in U$, $r_k \in R$ and $0 = t_0 \leq t_1 \leq t_2 \leq \ldots \leq t_n \leq T$
  ▪ Each interaction may be associated with a vector $f(u_k, r_k, t_k)$
  ▪ Time range of sequential interactions can be normalized to $[0, 1]$, thus we have $t_0 = t_1 = 0$ and $t_n = T = 1$.

• Intuition
  ▪ We represent vulnerabilities as edges such that the downstream task, i.e., link prediction, can predict the introduction of vulnerabilities
  ▪ The features of the nodes, i.e., users and repositories capture the properties that would influence the introduction of vulnerabilities, such as activity levels, cumulative number of vulnerabilities introduced at time $t$, programming language, etc. (Lazarine et al. 2020)
Research Design: Graph Formulation

- **Observable Graph**
  - Observable graph at time $t$ is the subgraph with edges, i.e., interactions, happened before time $t$.
  - The adjacency matrix of the observable graph at time $t \in (t_k, t_{k+1})$ is denoted by $A_k = \begin{bmatrix} 0 & B_k \\ B_k^T & 0 \end{bmatrix}$ where $B_k \in \mathbb{R} |U| \times |R|$ is the bi-adjacency matrix; element $B_{k,ur}$ denotes the number of interactions between $u$ and $r$ before time $t_{k+1}$, i.e.,
    \[ B_{k,ur} = \left| \left\{ (u', r', t') \in E | u' = u \land r' = r \land t' < t_{k+1} \right\} \right|. \]

- **Temporal Embedding of Interaction Graph**
  - The goal of temporal embedding is to learn a function $x : (U \cup R) \times [0,T] \rightarrow \mathbb{R}^d$ that reflects the continuous evolution of users and repositories over time.
  - $x(u, t) : d$-dimensional embeddings of user $u$ at time $t$
  - $x(r, t) : d$-dimensional embeddings of repository $r$ at time $t$
Research Design: Graph Formulation and Features

The selected features can be categorized by the element of the interaction that impacts the introduction of the vulnerability:

- **User-related features:** The experience, and previous activities of the developers, as well as the frequency and severity of the vulnerabilities introduced influence the propensity of a developer introducing a vulnerability. These factors are operationalized by considering the number of repositories owned, comment activities, a cumulative sum of the vulnerabilities and associated severity before the interaction.

- **Repository-related features:** Characteristics of the repositories, such as the language, its popularity, the number of developers collaborating on it, the number of pre-existing vulnerabilities, as well as pre-existing vulnerabilities and their nature influence the propensity of vulnerabilities being introduced to a repository. These factors are operationalized by the number of stars, open issues, comments, as well as the number and type of pre-existing vulnerabilities.

- **Interaction-related features:** The severity of the vulnerability would influence the propensity of vulnerability introduction. For instance, the incidence of low-severity vulnerabilities is higher than high-severity vulnerabilities.
# Research Design: Graph Formulation and Features

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature</th>
<th>Description</th>
<th>Rationale</th>
<th>Reference</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nodal Features: User</strong></td>
<td>Repositories Owned</td>
<td>The number of repositories owned by a user.</td>
<td>Captures developer experience</td>
<td>Bao et al. 2019</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Repositories Starred</td>
<td>The number of repositories starred by a user.</td>
<td>Captures developer activity</td>
<td>Bao et al. 2019</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Cumulative Comments</td>
<td>Cumulative number of comments made by a user.</td>
<td>Captures developer activity</td>
<td>Bao et al. 2019</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Public Repositories</td>
<td>The number of public repositories starred by a user.</td>
<td>Captures developer activity</td>
<td>Bao et al. 2019</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Cumulative Vulnerabilities</td>
<td>Cumulative vulnerabilities before interaction</td>
<td>Risk carried by developer</td>
<td>Lazarine et al. 2019</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Cumulative Severity Score</td>
<td>Cumulative severity score of vulnerabilities introduced</td>
<td>Risk severity carried by developer</td>
<td>Lazarine et al. 2019</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature</th>
<th>Description</th>
<th>Rationale</th>
<th>Reference</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nodal Features: Repositories</strong></td>
<td>Language</td>
<td>The primary language of the repository.</td>
<td>Vulnerabilities are language specific</td>
<td>Bao et al. 2019</td>
<td>Python</td>
</tr>
<tr>
<td></td>
<td>Fork</td>
<td>Whether the repository is forked.</td>
<td>Captures novel development</td>
<td>Bao et al. 2019</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Count of Open Issues</td>
<td>The number of open issues at the time of data collection.</td>
<td>Measure of collaboration and activity</td>
<td>Bao et al. 2019</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Stars</td>
<td>Number of stars that a repository has.</td>
<td>Measure of popularity</td>
<td>Bao et al. 2019</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Watches</td>
<td>Number of times a repository has been watched.</td>
<td>Measure of popularity</td>
<td>Bao et al. 2019</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Forks</td>
<td>Number of times a repository has been forked.</td>
<td>Measure of popularity</td>
<td>Bao et al. 2019</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Pull requests</td>
<td>Number of pull requests for the repository.</td>
<td>Measure of popularity</td>
<td>Bao et al. 2019</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Size</td>
<td>The size of the repository.</td>
<td>Captures functionality</td>
<td>Bao et al. 2019</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>Cumulative Vulnerabilities</td>
<td>Cumulative vulnerabilities before interaction</td>
<td>Risk carried by repository</td>
<td>Lazarine et al. 2019</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Cumulative Severity Score</td>
<td>Cumulative severity score of vulnerabilities introduced</td>
<td>Risk severity carried by repository</td>
<td>Lazarine et al. 2019</td>
<td>50</td>
</tr>
</tbody>
</table>

| **Edge Features** | Vulnerability Severity | Severity of vulnerability | Nature of vulnerability associated with specific commit | Lazarine et al. 2019 | 1 (one hot) |

Table 4. Features potentially affecting vulnerability introduction, categorized by graph components.
Proposed SeCoPE: Weighted Continuous Propagation Unit

- **GOAL:** What $X(t)(t \in t_k, t_{k+1})$ will be, given $X(t_k^+)$, before the next interaction

- Node representations at time $t$: $X(t) = X(t_k^+) + \int_{t_k}^{t} h \, dt$ where the function $h$ is a GNN-based function to model continuous propagation and evolution (CGNN)

- A (spectral radius) controls the extent of impact of the center node on its neighbors

\[
L_k = \alpha' \left( I + D_k^{-\frac{1}{2}} A_k D_k^{-\frac{1}{2}} \right)
\]

where $A_k$ is the adjacency matrix of the observable graph at time $(t \in t_k, t_{k+1})$, $D_k$ is the degree matrix and $\alpha \in (0, 1)$ is a parameter controlling the spectral radius of $L_k$

- Thus, we have the following GNN:

\[
\frac{d}{dt} X(t) = (L_k - I)X(t) + E, (t \in t_k, t_{k+1})
\]

- Intuition: Users have differential impact based on their relative position and influence in the network

- The knowledge, information and code from highly influential users would be referenced more: greater impact of influential developers on codebases, and influential codebases on developers

- The extent of influence on neighboring nodes can be captured using centrality measures

- Eigenvector centrality can capture the transitive influence/relative prestige score with respect to the entire network, reflecting the hierarchical structures within organizations
### Evaluation

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Justification</th>
<th>Methods</th>
<th>Description</th>
<th>References</th>
<th>Evaluation Metrics*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SeCoPE against RNN-based methods</td>
<td>Recurrent neural network models are commonly used to train nodal embeddings for link prediction</td>
<td>JODIE</td>
<td>Coupled recurrent neural network model; learns embedding trajectories of two types of nodes (e.g., users and repositories)</td>
<td>Kumar et al. 2019; Zhang et al. 2021</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CoPE</td>
<td>Modeling continuous propagation and evolution</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SeCoPE against Attention-based methods</td>
<td>Evaluate attention-based methods that account for neighboring nodes’ attributes</td>
<td>DyREP</td>
<td>Temporal attention layer to capture the neighbors’ interactions</td>
<td>Trivedi et al. 2019; Xu et al. 2020; Rossi et al. 2020</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TGN</td>
<td>“Memory” module to update node embeddings</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TGAT</td>
<td>Attention layer to efficiently aggregate temporal-topological neighborhood features</td>
<td></td>
</tr>
</tbody>
</table>

The gold standard dataset is the result of the vulnerability assessment (introduction of vulnerability).

Precision is an important metric because the costs of False Positive is high. In vulnerability introduction prediction, an interaction that will not introduce a vulnerability has been identified as vulnerability introduction.

F1-score is an important metric for comparing models, as it is not sensitive to data imbalance.

80% of the timestamps are used for training, 10% for validation, and 10% for testing.
## Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPERIMENT 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCAR</td>
<td>JODIE</td>
<td>0.569**</td>
<td>0.54**</td>
<td>0.634**</td>
<td>0.556**</td>
</tr>
<tr>
<td></td>
<td>CoPE</td>
<td>0.910**</td>
<td>0.657**</td>
<td>0.663**</td>
<td>0.660**</td>
</tr>
<tr>
<td></td>
<td>seCoPE</td>
<td>0.959</td>
<td>0.764</td>
<td>0.704</td>
<td>0.733</td>
</tr>
<tr>
<td>EXPERIMENT 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCAR</td>
<td>TGN</td>
<td>0.589**</td>
<td>0.562**</td>
<td>0.913</td>
<td>0.692**</td>
</tr>
<tr>
<td></td>
<td>DyREP</td>
<td>0.473**</td>
<td>0.413**</td>
<td>0.383**</td>
<td>0.33**</td>
</tr>
<tr>
<td></td>
<td>TGAT</td>
<td>0.585**</td>
<td>0.556**</td>
<td>0.884**</td>
<td>0.681**</td>
</tr>
<tr>
<td></td>
<td>seCoPE</td>
<td>0.959</td>
<td>0.764</td>
<td>0.704**</td>
<td>0.733</td>
</tr>
</tbody>
</table>

### Key Observations:
- seCoPE outperforms state-of-the-art RNN-based dynamic graph deep learning.
- We perform a one-sided t-test to check for statistical significance and compare each method with the best performing method, i.e., seCoPE for all metrics.
- 16.2% increase for precision for NCAR.

![Figure 7. Precision and F1-Score for Experiment 1](image)

![Figure 8. Precision and F1-Score for Experiment 2](image)
Case Study: Provision of Personalized and Timely Security Trainings

- To demonstrate the practical value of seCoPE, we conduct a case study illustrating the timely identification of high-risk actors it enables.

- The case study demonstrates how seCoPE can be utilized by various stakeholders to identify individual developers such that personalized trainings can be delivered.

- Personalized security trainings will better engage developers and reduce vulnerability incidence due to human error.

![Diagram](image)

**Figure 13. Case Study**
Case Study: Provision of Personalized and Timely Security Trainings

• For whom is the research significant?
  • Managed Security Service Provider (MSSP): Monitoring and management of security devices and systems is often outsourced to MSSP’s, who can provide personalized security awareness trainings.
  • Internal SOC: An information security team that monitors, detects and analyzes events on the network or system to prevent and resolve issues, and can conduct internal security awareness trainings.

• How is the research significant?
  • Instrumental benefits: Enabling stakeholders to identify high risk actors for targeted security awareness trainings, optimal assignment of developers to repositories, as input in developer scorecards.

• How is the research operationalized?
  The following steps can be performed by security analysts at MSSP’s or within organizations:
  1. Collect the GitHub repositories for the organization.
  2. Conduct a vulnerability assessment at the commit-level to obtain vulnerabilities introduced by each user.
  3. Generate an interaction graph.
  4. Train the model, using seCoPE to generate risk profiles (nodal embeddings) for users and repositories.
  5. Create user-repository pairs for varying periods of time. For any given developer and repository, the introduction of a vulnerability can be predicted over the given time period.
Case Study: Provision of Personalized and Timely Security Trainings

- To illustrate the identification of developers, we run the seCoPE model for the first 25% of the dataset. The predictions made by the model are then used to hypothesize about the subsequent training efficacy.
- We plot the number of low severity vulnerabilities (CWE-119/CWE-120/CWE-362/CWE-190) introduced by user “nief” into Cyverse repositories over time.
- We can see that within a category of vulnerabilities, vulnerability introduction can be used to predict future behavior.
- When run on the first 25% of the dataset, the seCoPE model is able to successfully predict the encircled vulnerability, i.e., commit #cd22f58.

![Figure 14. Vulnerability Introduction by User nief](image)
Case Study: Provision of Personalized and Timely Security Trainings

- CWE 119/120 can lead to potential memory overflows, CWE-362 can lead to attackers opening malicious files on the device, and CWE-120 can lead to buffer overflows when copying.

- If targeted security training is provided to user before time 97547292, the introduction of the vulnerabilities in the highlighted area can potentially be mitigated.

![Figure 15. Vulnerability Introduction by User nief](image-url)
Conclusion and Future Directions

Implications for practice:

- Reduce the incidence of vulnerabilities
- Efficiently utilize training resources
- Investment in human capital through workforce development

Several promising directions for future research:

- This design artifact can be deployed in a field experiment to compare the user perceptions and long-term efficacy of using targeted vs. non-targeted trainings.
- Future research can contextualize the proposed link prediction approach to identify awareness and training needs at different granularities, e.g., developers, teams, or departments, or in different contexts, e.g., technologies or projects.
- The predictive model can be improved by incorporating data available to the firm that could impact the propagation of knowledge, such as communication logs and prior trainings.
Thank you!

Questions or Comments?

__

Agrim Sachdeva
Kelley School of Business, Indiana University
agsach@iu.edu


References


