ICSD: An Automatic System for Insecure Code Snippet Detection in Stack Overflow over Heterogeneous Information Network

Yanfang Ye ✉, Shifu Hou, Lingwei Chen, Xin Li
Department of Computer Science
and Electrical Engineering
West Virginia University, WV, USA
yanfang.ye@mail.wvu.edu

Liang Zhao
Department of Information Science
and Technology
George Mason University, VA, USA
lzhao9@gmu.edu

Shouhuai Xu
Department of Computer Science
University of Texas at San Antonio, TX, USA
shxu@cs.utsa.edu

Jiabin Wang, Qi Xiong
Tencent Security Lab
Tencent, Guangdong, China
luciferwang@tencent.com

ABSTRACT

As the popularity of modern social coding paradigm such as Stack Overflow grows, its potential security risks increase as well (e.g., insecure codes could be easily embedded and distributed). To address this largely overlooked issue, in this paper, we bring an important new insight to exploit social coding properties in addition to code content for automatic detection of insecure code snippets in Stack Overflow. To determine if the given code snippets are insecure, we not only analyze the code content, but also utilize various kinds of relations among users, badges, questions, answers, code snippets and keywords in Stack Overflow. To model the rich semantic relationships, we first introduce a structured heterogeneous information network (HIN) for representation and then use meta-path based approach to incorporate higher-level semantics to build up relatedness over code snippets. Later, we propose a novel network embedding model named *snippet2vec* for representation learning in HIN where both the HIN structures and semantics are maximally preserved. After that, a multi-view fusion classifier is constructed for insecure code snippet detection. To the best of our knowledge, this is the first work utilizing both code content and social coding properties to address the code security issues in modern software coding platforms. Comprehensive experiments on the data collections from Stack Overflow are conducted to validate the effectiveness of the developed system *ICSD* which integrates our proposed method in insecure code snippet detection by comparisons with alternative approaches.

CCS CONCEPTS

• **Security and privacy → Software security engineering**; • **Computing methodologies → Machine learning algorithms**;

KEYWORDS

Social Coding, Code Security, Heterogeneous Information Network, Network Representation Learning, Multi-view Fusion

1 INTRODUCTION

Nowadays, as computing devices and Internet become increasingly ubiquitous, software has played a vital role in modern society covering many corners of our daily lives, such as Instant Message (IM) tools of WhatsApp and WeChat. In recent years, there has been an exponential growth in the number of software; it's estimated that the global software market reached approximately $406.6 billions in 2017 [30]. Unlike conventional approaches (e.g., code handbook based), modern software developers heavily engage in a social coding environment, i.e., they tend to reuse code snippets and libraries or adapt existing ready-to-use projects during the process of software development [45]. In particular, Stack Overflow [33], as the largest online programming discussion platform, has attracted 8.9 million registered developers [38]. The vibrant discussions and ready-to-use code snippets make it one of the most important information sources to software developers [10]. Despite the apparent benefits of such social coding environment, its profound implications into the security of software remain poorly understood [1, 17]. For example, *can one trust code snippets posted in Stack Overflow?*

As the popularity of Stack Overflow grows, the incentive of launching a large-scale security attack by exploiting the vulnerability of posted code snippets increases as well. According to a recent study [3], collected question-answer samples from Stack Overflow contain various security-related issues such as encryption with insecure mode, insecure Application Programming Interface (API) usage and so on. Those innocent-looking yet insecure code snippets - if not properly handled and directly transplanted to production software - could cause severe damage or even a disaster (e.g., disrupting system operations, leaking sensitive information) [3, 47]. For example, as shown in Figure 1, since cryptocurrency has grown popular, attackers have injected malicious mining code such as *Coinhive* - a cryptocurrency mining service - into Stack Overflow; once innocent developers reuse or copy-paste such code snippets to generate the production software, the software users' devices could be compromised (e.g., processing power would be stolen to mine bits of cryptocurrency). Stack Overflow has been aware of

ACSAC '18, December 3–7, 2018, San Juan, PR, USA

ACM ISBN 978-1-4503-6569-7/18/12…$15.00
https://doi.org/10.1145/3274694.3274742

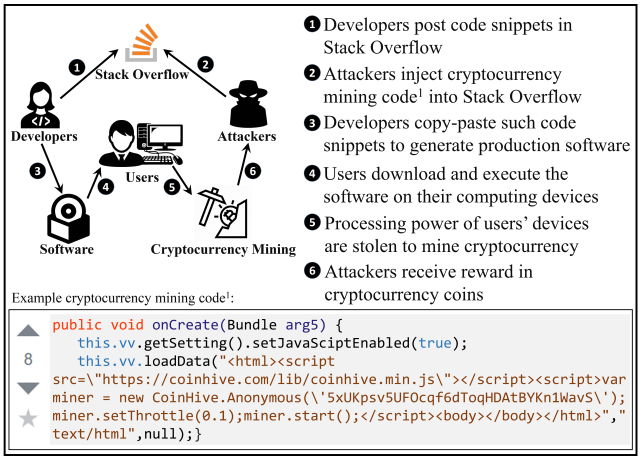

Figure 1: Example of code security attacks in Stack Overflow.

the negative impacts of insecure code infiltrations; unfortunately there has been no principled way of dealing with insecure code snippets included in the posted questions/answers other than labeling the moderator flag, downvoting those threads or warning in the comments [3]. Given the rich structure and information of Stack Overflow with ever-evolving programming languages, there is apparent and imminent need to develop novel and sound solutions to address the issue of code snippet security in Stack Overflow.

To address the above challenges, an important new insight brought by this work is to exploit social coding properties in addition to code content for automatic detection of insecure code snippets. As a social coding environment, Stack Overflow is characterized by user communication through questions and answers [12], that is, a rich source of heterogeneous information are available in Stack Overflow including users, badges, questions, answers, code snippets, and the rich semantic relations among them. For example, as shown in Figure 2, to detect if a code snippet (*Code-2*) is insecure, using the code content (e.g., methods, functions, APIs, etc.) alone may not be sufficient; however, other rich information provided in Stack Overflow could be valuable for the prediction, such as (1) the same user (*User-1*) may be prone to post different insecure code snippets (*Code-1* and *Code-2*) due to his/her coherent code writing style, or (2) similar insecure code snippets (*Code-2* and *Code-3*) may be posted by a group of inexperienced users (*User-1* and *User-2* both only had the bronze badge of "commentator" that could be gained by leaving 10 comments in Stack Overflow).

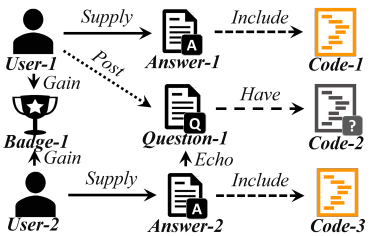

Figure 2: An example of relatedness over code snippets.

To utilize the social coding properties of Stack Overflow data (i.e., including different entities of users, badges, questions, answers and code snippets, as well as the rich semantic relationships among them) in addition to code content (i.e., keywords extracted from code snippets such as function names, methods and APIs), in this paper, we propose to introduce a heterogeneous information network (HIN) [39, 41] as an abstract representation. Then we use meta-path [41] to incorporate higher-level semantic relationships to build up relatedness over the code snippets. Afterwards, to reduce the high computation and space cost, we further propose a novel network embedding model named *snippet2vec* for node (i.e., code snippet) representation learning in the HIN, where both HIN structure and semantics are maximally preserved. After that, a multi-view fusion classifier is constructed for automatic detection of insecure code snippets in Stack Overflow. We develop a system called *ICSD* integrating our proposed method for insecure code snippet detection, which has the following major traits:

- ***Novel feature representation of Stack Overflow data.*** Security risks arising from the new paradigm of social coding are more sophisticated than those from conventional wisdom, which requires a deeper understanding and a greater modeling effort. In addition to code content, a rich source of heterogeneous information in Stack Overflow including users, badges, questions, answers, code snippets, and the semantic relations among them is also available. To utilize such social coding properties (e.g., *question-code*, *answer-code*, *code-keywords*, *user-question*, *user-answer*, *question-answer*, and *user-badge* relations), we propose to introduce HIN as an abstract representation of Stack Overflow data. Then a meta-path based approach is exploited to characterize the relatedness over code snippets. The proposed solution provides a natural way of expressing complex relationships in social coding platforms such as Stack Overflow, which has not been studied in the open literature to our best knowledge.
- ***Multi-view fusion classifier based on novel representation learning model.*** Based on a set of built meta-path schemes, to reduce the high computation and space cost, a new network embedding model named *snippet2vec* is proposed to learn the low-dimensional representations for the nodes (i.e., code snippets) in the HIN, which are capable to preserve both the semantics and structural correlations between different types of nodes. Then, given different sets of meta-path schemes, different kinds of node (i.e., code snippet) representations will be learned by using *snippet2vec*. To aggregate these different learned node representations, we propose a multi-view fusion classifier to learn importance of them and thus to make predictions (i.e., a given code snippet will be labeled as either insecure or not).
- ***A practical system for automatic detection of insecure code snippets.*** Based on the collected and annotated data from Stack Overflow, we develop a practical system named *ICSD* integrating our proposed method for automatic detection of insecure code snippets. We provide comprehensive experimental studies to validate the performance of our developed system in comparisons with alternative approaches. This work is the first attempt utilizing both code content and social coding properties for automatic analysis of code security in Stack Overflow. The proposed method and developed system can also be easily expanded to

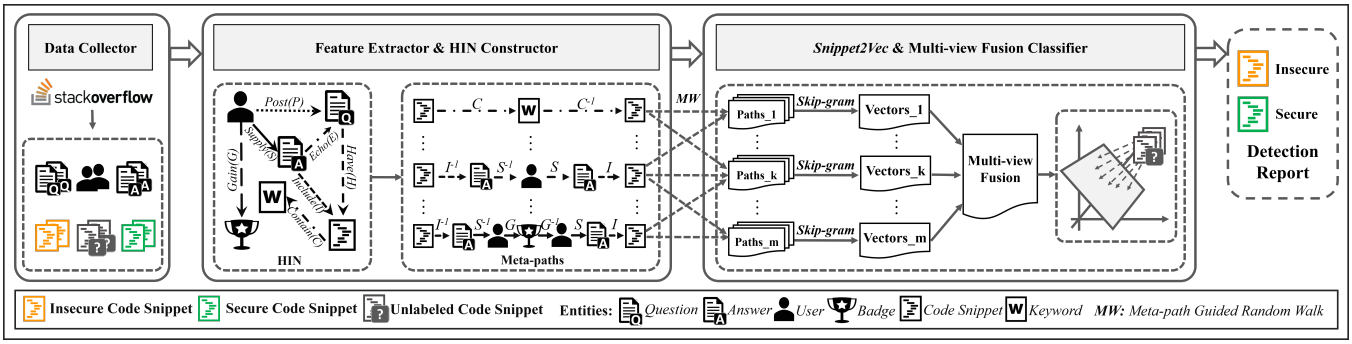

Figure 3: System architecture of *ICSD*.

code security analysis in other social coding platforms, such as GitHub and Stack Exchange.

The rest of the paper is organized as follows. Section 2 presents our system architecture. Section 3 introduces our proposed method in detail. Based on the collected and annotated data from Stack Overflow, Section 4 systematically evaluates the effectiveness of our developed system *ICSD* integrating our proposed method by comparisons with alternative approaches. Section 5 discusses the related work. Finally, Section 6 concludes.

2 SYSTEM ARCHITECTURE

The system architecture of *ICSD* is shown in Figure 3, which is developed for insecure code snippet detection in Stack Overflow. It consists of the following major components:

- ***Data collector.*** A set of crawling tools are developed to collect the data from Stack Overflow. The collected data includes users' profiles, their posted questions and answers, and the code snippets embedded in the questions/answers.
- ***Feature extractor.*** Resting on the data collected from the previous module, to depict the code snippets, it first extracts the content-based features from the collected code snippets (i.e., keywords such as function names, methods and APIs), and then analyzes various relationships among different types of entities (i.e., user, badge, question, answer, code snippet, keyword), including i) *question-have-code*, ii) *answer-include-code*, iii) *code-contain-keyword*, iv) *user-post-question*, v) *user-supply-answer*, vi) *answer-echo-question*, and vii) *user-gain-badge* relations. (See Section 3.1 for details.)
- ***HIN constructor.*** In this module, based on the features extracted from the previous component, a structured HIN is first presented to model the relationships among different types of entities; and then different meta-paths are built from the HIN to capture the relatedness over code snippets from different views (i.e., with different semantic meanings). (See Section 3.2 for details.)
- ***snippet2vec.*** Based on the built meta-path schemes, to reduce the high computation and space cost, a new network embedding model *snippet2vec* is proposed to learn the low-dimensional representations for the nodes in HIN, which are capable to preserve both the semantics and structural correlations between different types of nodes. In *snippet2vec*, given a set of different meta-path schemes, a meta-path guided random walk strategy is first proposed to map the word-context concept in a text corpus into a HIN; then skip-gram is leveraged to learn effective node representation for a HIN. (See Section 3.3 for details.)
- ***Multi-view fusion classifier.*** Given different sets of meta-path schemes, different kinds of node (i.e., code snippet) representations will be learned by using *snippet2vec*. To aggregate these different representations, a multi-view fusion classifier is constructed to learn importance of them and thus to make predictions (i.e., the unlabeled code snippets will be predicted if they are insecure or not). (See Section 3.4 for details.)

3 PROPOSED METHOD

In this section, we present the detailed approaches of how we represent the code snippets in Stack Overflow utilizing both code content and social coding properties simultaneously, and how we solve the insecure code snippet detection problem based on the representation.

3.1 Feature Extraction

Code snippets. Stack Overflow provides the discussion platform for software developers to post their questions and answers about ever-evolving programming languages including Java, JavaScript, C/C++/C#, Python, PHP, perl, etc. In this paper, we will focus on Java programming language for Android application (app) development as a showcase for the following reasons: (1) Java is one of the most popular programming languages in Stack Overflow [44]. (2) Due to the mobility and ever expanding capabilities, mobile devices have recently surpassed desktop and other media - it is estimated that 77.7% of all devices connected to the Internet will be smart phones in 2019 [21, 22] (leaving PCs falling behind at 4.8%). Android, as an open source and customizable operating system for mobile devices, is currently dominating the smart phone market by 82.8% [24]. (3) Billions of mobile device users with millions of Android apps installed have attracted more and more developers; however, most of these Android mobile apps have poorly implemented security mechanisms partially because developers are inexperienced, distracted or overwhelmed [1, 35]. Indeed developers tend to request more permissions than what are actually needed, often use insecure options for Inter Component Communication (ICC), and fail to store sensitive information in private areas [44]. Code snippets in Stack Overflow are surrounded by ⟨*code*⟩ ⟨/*code*⟩ tags, and

they can thus easily be separated from accompanying texts before being extracted. Then, content-based features will be further extracted from the collected code snippets: we will first remove all the punctuations and stopwords; and then we will extract the keywords including function names, methods, APIs and parameters to represent the content of code snippets.

Social coding properties. To depict a code snippet in Stack Overflow, we not only utilize its above extracted content-based features, but also consider its social coding properties including followings.

- ***R1***: To describe the relation that a question thread has a code snippet included, we generate the ***question-have-code*** matrix **H** where each element $h_{i,j} \in \{0, 1\}$ indicates whether question i has code snippet j.
- ***R2***: To denote the relation that an answer thread includes a code snippet, we generate the ***answer-include-code*** matrix **I** where each element $i_{i,j} \in \{0, 1\}$ means if answer i includes code snippet j.
- ***R3***: To represent the relation that a code snippet contains a specific keyword (e.g., function name of "Coinhive"), we build the ***code-contain-keyword*** matrix **C** whose element $c_{i,j} \in \{0, 1\}$ denotes whether code snippet i contains keyword j.
- ***R4***: To describe the relation between a user and a question he/she posts, we generate the ***user-post-question*** matrix **P** where each element $p_{i,j} \in \{0, 1\}$ denotes if user i posts question j.
- ***R5***: To represent the relation of a user and an answer he/she supplies, we generate the ***user-supply-answer*** matrix **S** where each element $s_{i,j} \in \{0, 1\}$ denotes whether the user i supplies answer j.
- ***R6***: To denote the Q&A relationship, we build the ***answer-echo-question*** matrix **E** whose element $e_{i,j} \in \{0, 1\}$ denotes whether answer i echoes/responds to question j.
- ***R7***: In order to encourage engagement, Stack Overflow has adopted a strategy of *gamification* [12] - users will be rewarded for their valued contributions to the forum. For example, "illuminator" badge (gold level in answer badges) will be awarded to the users who edit and answer 500 questions (both actions within 12 hours, answer score > 0). This can be seen as a measure of a user's expertise by potential recruiters [6]. In Stack Overflow, there are different kinds of badges (e.g., question badges, answer badges, etc.) with different levels (i.e., gold, silver, and bronze). To describe the relationship between a user and a specific badge he/she gains, we build the ***user-gain-badge*** matrix **G** whose element $g_{i,j} \in \{0, 1\}$ denotes if user i gain badge j.

3.2 HIN Constructor

In order to depict users, badges, questions, answers, code snippets, keywords as well as the rich relationships among them (i.e., ***R1***-***R7***), it is important to model them in a proper way so that different kinds of relations can be better and easier handled. We introduce how to use HIN, which is capable to be composed of different types of entities and relations, to represent the code snippets in Stack Overflow by using the features extracted above. We first present the concepts related to HIN as follows.

Definition 3.1. ***Heterogeneous information network (HIN)*** [40]. A HIN is defined as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with an entity type mapping $\phi: \mathcal{V} \rightarrow \mathcal{A}$ and a relation type mapping $\psi: \mathcal{E} \rightarrow \mathcal{R}$, where $\mathcal{V}$ denotes the entity set and $\mathcal{E}$ is the relation set, $\mathcal{A}$ denotes the entity type set and $\mathcal{R}$ is the relation type set, and the number of entity types $|\mathcal{A}| > 1$ or the number of relation types $|\mathcal{R}| > 1$. The ***network schema*** [40] for a HIN $\mathcal{G}$, denoted as $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, is a graph with nodes as entity types from $\mathcal{A}$ and edges as relation types from $\mathcal{R}$.

HIN not only provides the network structure of the data associations, but also provides a high-level abstraction of the categorical association. For our case, i.e., the detection of insecure code snippets in Stack Overflow, we have six entity types (i.e., user, badge, question, answer, code snippet, keyword) and seven types of relations among them (i.e., ***R1***-***R7***). Based on the definitions above, the network schema for HIN in our application is shown in Figure 4, which enables the code snippets in Stack Overflow to be represented in a comprehensive way that utilizes both their content-based information and social coding properties.

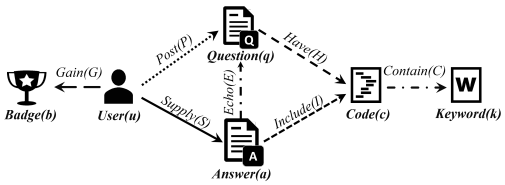

Figure 4: Network schema for HIN in our application.

The different types of entities and relations motivate us to use a machine-readable representation to enrich the semantics of relatedness among code snippets in Stack Overflow. To handle this, the concept of meta-path has been proposed [41] to formulate the higher-order relationships among entities in HIN. Here, we follow this concept and extend it to our application of insecure code snippet detection in Stack Overflow.

Definition 3.2. ***Meta-path*** [41]. A meta-path $\mathcal{P}$ is a path defined on the graph of network schema $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, and is denoted in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \ldots \xrightarrow{R_L} A_{L+1}$, which defines a composite relation $R = R_1 \cdot R_2 \cdot \ldots \cdot R_L$ between types A_1 and A_{L+1}, where $\cdot$ denotes relation composition operator, and L is the length of $\mathcal{P}$.

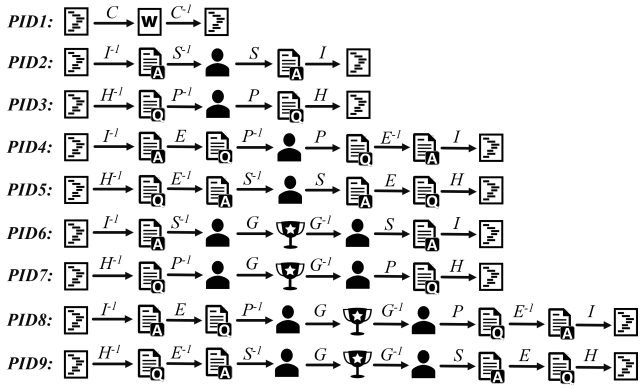

Figure 5: Meta-paths built for insecure code snippet detection (The symbols are the abbreviations shown in Figure 4).

Given a network schema with different types of entities and relations, we can enumerate a lot of meta-paths. In our application, based on the collected data, resting on the seven different kinds of relationships, we design nine meaningful meta-paths for characterizing relatedness over code snippets in Stack Overflow, i.e., ***PID1-PID9*** shown in Figure 5. Different meta-paths depict the relatedness between two code snippets at different views. For example, the meta-path ***PID2*** formulates the relatedness over code snippets in Stack Overflow: $code \xrightarrow{Include^{-1}} answer \xrightarrow{Supply^{-1}} user \xrightarrow{Supply} answer \xrightarrow{Include} code$ which means that two code snippets can be connected as they are included in the answers supplied by the same user; while another meta-path ***PID6***: $code \xrightarrow{Include^{-1}} answer \xrightarrow{Supply^{-1}} user \xrightarrow{Gain} reputation \xrightarrow{Gain^{-1}} user \xrightarrow{Supply} answer \xrightarrow{Include} code$ denotes that two code snippets are related as they are included in the answers supplied by the users with the same kind of badge (e.g., "illuminator" badge) indicating similar expertise or contribution. In our application, meta-path is a straightforward method to connect code snippets via different relationships among different entities in HIN, and enables us to depict the relatedness over code snippets in Stack Overflow utilizing both their content-based information and social coding properties in a comprehensive way.

3.3 *snippet2vec*: HIN Representation Learning

To measure the relatedness over HIN entities (e.g., code snippets), traditional representation learning for HIN [20, 41, 46, 48] mainly focuses on factorizing the matrix (e.g., adjacency matrix) of a HIN to generate latent-dimension features for the nodes (e.g., code snippets) in the HIN. However, the computational cost of decomposing a large-scale matrix is usually very expensive, and also suffers from its statistical performance drawback [19]. To reduce the high computation and space cost, it calls for scalable representation learning method for HIN. Given a HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the ***HIN representation learning*** task [13, 18] is to learn a function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ that maps each node $v \in \mathcal{V}$ to a vector in a d-dimensional space $\mathbb{R}^d$, $d \ll |\mathcal{V}|$ that are capable to preserve the structural and semantic relations among them.

To solve the problem of HIN representation learning, due to the heterogeneous property of HIN (i.e., network consisting of multi-typed entities and relations), it is difficult to directly apply the conventional homogeneous network embedding techniques (e.g., DeepWalk [34], LINE [43], node2vec [19]) to learn the latent representations for HIN. To address this issue, HIN embedding methods such as metapath2vec [13] was proposed. In metapath2vec, given a meta-path scheme, it employs meta-path based random walk and heterogeneous skip-gram to learn the latent representations for HIN such that the semantic and structural correlations between different types of nodes could be persevered. The metapath2vec was proposed to support one meta-path scheme to guide the walker traversing HIN; however, in our application, the code snippets in Stack Overflow can be connected through nine different meta-path schemes. It may not be feasible to directly employ metapath2vec in our case for insecure code snippet detection. To put this into perspective, as shown in Figure 6, we gain further insight into Stack Overflow data and have following interesting findings:

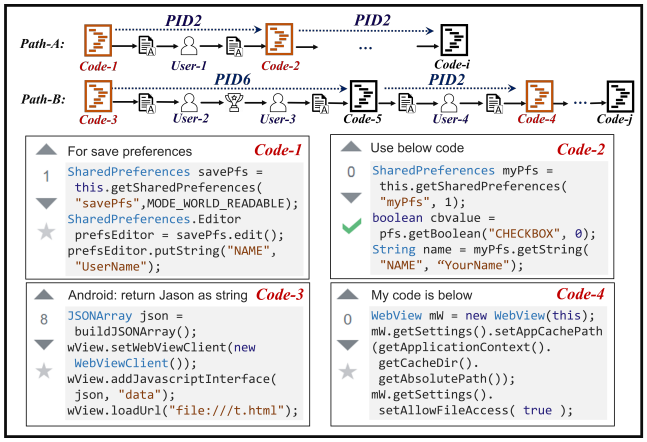

Figure 6: Random walk guided by single meta-path vs. random walk guided by multiple meta-paths.

- ***Finding 1:*** Both insecure *Code-1* and *Code-2* (i.e., they can both cause potential confidential information leakage) are posted by *User-1* "Ke***a" (we here anonymize his user name) answering the questions about string access for Android app. Actually, *Code-1* and *Code-2* can be connected by the *Path-A* guided by the designed meta-path *PID2*.
- ***Finding 2:*** The insecure codes of *Code-3* (i.e., it may allow users to remotely execute the malicious code) and *Code-4* (i.e., it can cause potential data breach) are connected in the way that (1) *Code-3* and *Code-5* are related as they were posted by *User-2* and *User-3* who only had the bronze badge of "student" (i.e., first question with score of 1 or more); and then (2) *User-4* copied and pasted *Code-5* while also provided *Code-4* to answer another user's posted question. In this way, *Code-3* and *Code-4* can be connected by the *Path-B* guided by meta-paths of *PID6* and *PID2*.

Based on the above observations, metapath2vec [13] fails to generate the path such as *Path-B* to represent the relatedness between code snippets like *Code-3* and *Code-4*. To address this issue, we design a new network embedding model *snippet2vec* to learn desirable node representations in HIN: first, a new random walk method guided by different meta-paths is proposed to map the word-context concept in a text corpus into a HIN; then skip-gram is leveraged to learn effective node representation for a HIN.

Random walk guided by different meta-paths. Given a source node v_j in a homogeneous network, the traditional random walk is a stochastic process with random variables $v_j^1, v_j^2, ..., v_j^k$ such that v_j^{k+1} is a node chosen at random from the neighbors of node v_k. The transition probability $p(v_j^{i+1}|v_j^i)$ at step i is the normalized probability distributed over the neighbors of v_j^i by ignoring their node types. However, this mechanism is unable to capture the semantic and structural correlations among different types of nodes in a HIN. Here, we show how we use different built meta-paths to guide the random walker in a HIN to generate the paths of multiple types of nodes. Given a HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with schema $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$, and a set of different meta-paths $\mathcal{S} = \{\mathcal{P}_j\}_{j=1}^n$ (e.g., in *Finding2*, $\mathcal{S} = \{PID6, PID2\}$), each of which is in the form of

$A_1 \rightarrow ...A_t \rightarrow A_{t+1}... \rightarrow A_l$, we put a random walker to traverse the HIN. The random walker will first randomly choose a meta-path $\mathcal{P}_k$ from $\mathcal{S}$ and the transition probabilities at step i are defined as follows:

$$p(v^{i+1}|v^i_{A_t}, \mathcal{S}) = \begin{cases} \frac{\lambda}{|\mathcal{S}|}\frac{1}{|N_{A_{t+1}}(v^i_{A_t})|} & \text{if } (v^{i+1}, v^i_{A_t}) \in \mathcal{E}, \phi(v^i_{A_t}) = A_c, \phi(v^{i+1}) = A_{t+1} \\ \frac{1}{|N_{A_{t+1}}(v^i_{A_t})|} & \text{if } (v^{i+1}, v^i_{A_t}) \in \mathcal{E}, \phi(v^i_{A_t}) \neq A_c, \\ & \phi(v^{i+1}) = A_{t+1}, (A_t, A_{t+1}) \in \mathcal{P}_k \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where ϕ is the node type mapping function, $N_{A_{t+1}}(v^i_{A_t})$ denotes the A_{t+1} type of neighborhood of node $v^i_{A_t}$, A_c is entity type of *Code*, and λ is the number of meta-paths starting with $A_c \rightarrow A_{t+1}$ in the given meta-path set $\mathcal{S}$. The walk paths generated by the above strategy are able to preserve both the semantic and structural relations between different types of nodes in the HIN, and thus will facilitate the transformation of HIN structures into skip-gram.

Skip-gram. After mapping the word-context concept in a text corpus into a HIN via meta-path guided random walk strategy (i.e., a sentence in the corpus corresponds to a sampled path and a word corresponds to a node), skip-gram [31, 34] is then applied on the paths to minimize the loss of observing a node's neighbourhood (within a window w) conditioned on its current representation. The objective function of skip-gram is:

$$\arg\min_{Y} \sum_{-w \leq k \leq w, j \neq k} -\log p(v_{j+k}|Y(v_j)), \tag{2}$$

where $Y(v_j)$ is the current representation vector of v_j, $p(v_{j+k}|Y(v_j))$ is defined using the softmax function:

$$p(v_{j+k}|Y(v_j)) = \frac{\exp(Y(v_{j+k}) \cdot Y(v_j))}{\sum_{q=1}^{|\mathcal{V}|} \exp(Y(v_q) \cdot Y(v_j))}. \tag{3}$$

Due to its efficiency, we first apply hierarchical softmax technique [32] to solve Eq. 3; then the stochastic gradient descent [4] is employed to train the skip-gram.

3.4 Multi-view Fusion Classifier

Given a set of different meta-path schemes, by using the above proposed *snippet2vec*, the node (i.e., code snippet) representations will be learned in the HIN. In our application, as described in Section 3.2, we have nine meta-paths (i.e., *PID1–MID9*) which characterize the relatedness over code snippets at different views (i.e., with different semantic meanings). Based on our observations on the Stack Overflow data and leveraging the domain expertise, we generate m sets of meta-path schemes $\mathbf{S} = \{\mathcal{S}_i\}_{i=1}^m$ for *snippet2vec* to learn the node representations in the HIN, where $m = 4$ and $\mathbf{S} = \{(PID1, PID2, PID6), (PID1, PID3, PID7), (PID1, PID4, PID8), (PID1, PID5, PID9)\}$. Given these different sets of meta-paths, using *snippet2vec*, different node representations will be learned in the HIN. Here, we propose to use multi-view fusion to aggregate these different learned node representations for code snippet classification.

Given m kinds of node representations $Y_i (i = 1, ..., m)$ learned based on m sets of meta-path schemes, the incorporated node representations can be denoted as: $Y' = \sum_{i=1}^m (\alpha_i \times Y_i)$, where α_i $(i = 1, ..., m)$ is the weight of Y_i. To determine the weight of α_i for each mapped low-dimensional vector space Y_i, we measure the geometric distances among them. The distance measure based on the principal angles between two vector spaces is significant if and only if the vector spaces have the same dimensions [49]. In our case, the m mapped vector spaces are all with the same dimensions of d. Therefore, we apply the geodesic distance based on principal angles [25] to measure the geometric distances between the mapped vector spaces. The principal angle between space Y_i and Y_j is defined as the number $0 \leq \theta \leq \frac{\pi}{2}$ that satisfies:

$$\cos\theta = \max_{\mathbf{y} \in Y_i, \mathbf{y}' \in Y_j} \mathbf{y}^T \mathbf{y}'. \tag{4}$$

The angle θ is 0 if and only if $Y_i \bigcap Y_j \neq 0$, while $\theta = \frac{\pi}{2}$ if and only if $Y_i \perp Y_j$. Let $\theta_1, \theta_2, ..., \theta_d$ be the d principal angles between space Y_i and Y_j, the geodesic distance between them is formulated as:

$$d(Y_i, Y_j) = \sqrt{\theta_1^2 + \theta_2^2 + ... + \theta_d^2}. \tag{5}$$

Thus, we compute α_i for each mapped vector space Y_i as:

$$\alpha_i = \frac{\sum_{j=1, i \neq j}^m d(Y_i, Y_j)}{\sum_{i=1}^m \sum_{j=1, i \neq j}^m d(Y_i, Y_j)}. \tag{6}$$

To this end, the incorporated node representations Y' will be fed to the Support Vector Machine (SVM) to train the classification model, based on which the unlabeled code snippets can be predicted if they are insecure or not. Algorithm 1 shows the implementation of the our developed insecure code snippet detection system *ICSD*.

Algorithm 1: ***ICSD*** – Insecure code snippet detection in Stack Overflow based on structured HIN.

Input: The HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, network schema $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$, m sets of meta-path schemes $\mathbf{S} = \{\mathcal{S}_i\}_{i=1}^m$, number of walk paths per node r, walk length l, and vector dimension d, traning data set D_t, testing data set D_e

Output: $\mathbf{f}$: The labels for the testing code snippets.

```
for i = 1 → m do
    for j = 1 → |V| do
        for k = 1 → r do
            get l-length random walks guided by S_i (Eq. 1);
        end
    end
    Generate Y_i ∈ R^d using skip-gram (Eq. 2) ;
end
for i = 1 → m do
    Calculate α_i for Y_i using Eq. 4–Eq. 6 ;
end
Get incorporated node representations Y' = Σ_{i=1}^m (α_i × Y_i) ;
Train SVM using Y'_{D_t} ;
for n = 1 → |D_e| do
    Generate the label f_n using trained SVM ;
end
return f;
```

4 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we conduct four sets of experimental studies using the data collected from Stack Overflow to fully evaluate the performance of our developed system *ICSD* which integrates the above proposed method in insecure code snippet detection.

4.1 Experimental Setup

We develop a set of crawling tools to collect the data from Stack Overflow. As stated in Section 3.1, we consider Java programming language for Android app as a case study to evaluate our developed system. Note that it's also applicable to other kinds of programming languages in Stack Overflow. We use our developed crawling tools to collect users' profiles, question threads, answer threads, and code snippets in Stack Overflow in a period of time. By the date, we have collected 429,523 question threats and 623,746 answer threats posted by 213,560 users including 737,215 code snippets, through March 2010 to May 2018. To obtain the ground truth for the evaluation of different detection methods, we need to prelabel a fraction of code snippets (i.e., either secure or insecure). We first categorize code security risks and vulnerabilities for Android apps into six categories: (1) Android Manifest configuration, (2) WebView component, (3) data security, (4) file directory traversal, (5) implicit intents, and (6) security checking; and then we leverage our domain expertise and follow the principles such as least permission request, correct usage of HTTPS and TLS for networking, secure inter-component communication, secure storage to manually label a filtered set of 20,137 code snippets (i.e., 9,054 code snippets are labeled as *insecure* while 11,083 are *secure*). After feature extraction and based on the designed network schema, the constructed HIN has 80,405 nodes (i.e., 20,137 nodes with type of code snippet, 24,286 nodes with type of answer, 13,924 nodes with type of question, 21,471 with type of user, 94 with type of badges, and 493 with type of selected keywords) and 592,082 edges including relations of *R1-R7*. We use the performance indices shown in Table 1 to quantitatively validate the effectiveness of different methods in insecure code snippet detection.

Table 1: Performance indices of code snippet detection

Indices	Description
TP	# of code snippets correctly classified as insecure
TN	# of code snippets correctly classified as secure
FP	# of code snippets mistakenly classified as insecure
FN	# of insecure mistakenly classified as secure
Precision	$TP/(TP+FP)$
Recall/TPR	$TP/(TP+FN)$
ACC	$(TP+TN)/(TP+TN+FP+FN)$
F1	$2\times Precision\times Recall/(Precision+Recall)$

4.2 *snippet2vec* based on Different Sets of Meta-path Schemes

In this set of experiments, based on the dataset described in Section 4.1, we first evaluate the performance of different kinds of relatedness over code snippets depicted by different sets of meta-path schemes. In the experiments, given a specific set of meta-path schemes, we use *snippet2vec* to learn the latent representations of the nodes (i.e., code snippets) in the HIN, which are then fed to SVM to build the classification model for insecure code snippet detection. For SVM, we use LibSVM and the penalty is empirically set to be 10 while other parameters are set by default. As described in Section 3.4, we generate four sets of meta-path schemes (denoted as $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$) for *snippet2vec* to learn the node representations in the HIN. We conduct 10-fold cross validations for evaluation. The performances of four different sets of meta-path schemes (i.e., $\mathcal{S}_1$-$\mathcal{S}_4$) in comparison with nine individual meta-paths (i.e., *PID1–PID9*) in insecure code snippet detection are shown in Table 2.

Table 2: Detection Results of different meta-paths

ID	Meta-paths included	Precision	Recall	ACC	F1
$\mathcal{S}_1$	(PID1,PID2,PID6)	0.9065	0.8887	0.8883	0.8975
$\mathcal{S}_2$	(PID1,PID3,PID7)	0.8899	0.8678	0.8682	0.8787
$\mathcal{S}_3$	(PID1,PID4,PID8)	0.9028	0.8834	0.8834	0.8930
$\mathcal{S}_4$	(PID1,PID5,PID9)	0.8922	0.8709	0.8710	0.8814
$\mathcal{S}'_5$	(PID1)	0.8795	0.8561	0.8562	0.8676
$\mathcal{S}'_6$	(PID2)	0.8340	0.7988	0.8018	0.8160
$\mathcal{S}'_7$	(PID3)	0.8017	0.7657	0.7668	0.7833
$\mathcal{S}'_8$	(PID4)	0.8463	0.8179	0.8180	0.8318
$\mathcal{S}'_9$	(PID5)	0.8312	0.8001	0.8006	0.8153
$\mathcal{S}'_{10}$	(PID6)	0.8449	0.8119	0.8145	0.8281
$\mathcal{S}'_{11}$	(PID7)	0.8108	0.7708	0.7748	0.7903
$\mathcal{S}'_{12}$	(PID8)	0.8020	0.7642	0.7664	0.7826
$\mathcal{S}'_{13}$	(PID9)	0.7897	0.7518	0.7532	0.7703

From Table 2, we can see that different sets of meta-path schemes indeed show different performances in insecure code snippet detection, since each of them represents specific semantics in insecure code snippet detection. We also observe that: (1) *PID1* outperforms the other individual meta-paths (i.e., *PID2–PID9*), which indicates that the semantics of this meta-path reflect the problem of insecure code snippet detection better than the others. (2) The meta-paths of *PID2*, *PID4*, *PID6*, and *PID8* perform better than *PID3*, *PID5*, *PID7*, and *PID9* respectively; the reason behind this is that the code snippets posted in the answer threads are more likely to be reused by the developers than the ones posted in question threads, and thus they have closer connections. (3) Obviously, $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$ utilizing different meta-paths built from HIN are more expressive than each individual meta-path (i.e., *PID1–PID9*) in depicting the code snippets in Stack Overflow and thus achieve better detection performance. It will be interested to see the detection performance if different sets of meta-paths are further aggregated. This will be evaluated in the next set of experiments.

4.3 Comparisons with Different Network Representation Learning Models

In this set of experiments, we evaluate our developed system *ICSD* integrating our proposed method described in Section 3 by comparisons with several network representation learning methods: (1)

Table 3: Comparisons with other network representation learning methods in insecure code snippet detection

Metric	Method	10%	20%	30%	40%	50%	60%	70%	80%	90%
ACC	DeepWalk	0.6085	0.6409	0.6550	0.6674	0.6810	0.6958	0.7148	0.7269	0.7279
	LINE	0.6347	0.6559	0.6847	0.7075	0.7268	0.7364	0.7475	0.7635	0.7732
	metapath2vec	0.7772	0.7839	0.8197	0.8366	0.8490	0.8522	0.8663	0.8782	0.8826
	ICSD	0.7973	0.8133	0.8384	0.8566	0.8771	0.8835	0.8953	0.9068	0.9123
F1	DeepWalk	0.6308	0.6618	0.6764	0.6875	0.7006	0.7159	0.7329	0.7448	0.7461
	LINE	0.6569	0.6762	0.7047	0.7261	0.7451	0.7547	0.7644	0.7798	0.7892
	metapath2vec	0.7932	0.7990	0.8332	0.8493	0.8609	0.8633	0.8765	0.8878	0.8921
	ICSD	0.8121	0.8270	0.8508	0.8680	0.8871	0.8930	0.9036	0.9146	0.9197

DeepWalk [34] and LINE [43] which are homogeneous network embedding methods; and (2) metapath2vec [13] which is a HIN embedding model. For DeepWalk and LINE, we ignore the heterogeneous property of HIN and directly feed the HIN for representation learning; in metapath2vec, a walk path will be generated only based on a single meta-path scheme; while in our proposed *snippet2vec*, a walk path will be guided by a set of different meta-path schemes. The parameter settings used for *snippet2vec* are in line with typical values used for the baselines: vector dimension $d = 200$ (LINE: 200 for each order (1st- and 2nd-order)), walks per node $r = 10$, walk length $l = 80$, and window size $w = 10$. To facilitate the comparisons, we use the experimental procedure as in [13, 34, 43]: we randomly select a portion of labeled code snippets described in Section 4.1 (ranging from 10% to 90%) for training and the remaining ones for testing. For all the baselines, the SVM is used as the classification model; for *ICSD*, based on the four given sets of meta-path schemes, it will generate four different kinds of node representations using *snippet2vec* and then use multi-view fusion classifier proposed in Section 3.4 to train the classification model. Table 3 illustrates the detection results of different network representation learning models. From Table 3, we can see that *ICSD* integrating the proposed *snippet2vec* model consistently and significantly outperforms all baselines for insecure code snippet detection in terms of *ACC* and *F1*. That is to say, *snippet2vec* learns significantly better code snippet representation than current state-of-the-art methods. The success of *snippet2vec* lies in the proper consideration and accommodation of the heterogeneous property of HIN (i.e., the multiple types of nodes and relations), and the advantage of random walk guided by different meta-paths for sampling the node paths. Furthermore, from Table 2 and Table 3, we can also observe that using the multi-view fusion classifier proposed in Section 3.4 to aggregate different node representations learned based on different sets of meta-graph schemes can significantly improve the detection performance.

4.4 Comparisons with Traditional Machine Learning Methods

In this set of experiments, based on the dataset described in Section 4.1, we compare *ICSD* which integrates our proposed method with other traditional machine learning methods by 10-fold cross validations. For these methods, we construct three types of features: ***f–1***: content-based features (i.e., keywords extracted from code snippets described in Section 3.1); ***f–2***: two relation-based features associated with code snippets (i.e., ***R1*** and ***R2*** introduced in Section 3.1); ***f–3***: augmented features of content-based features and ***R1–R2***. Based on these features, we consider two typical classification models, i.e., Naive Bayes (NB) and SVM. The experimental results are illustrated in Table 4. From the results we can observe that feature engineering (***f-3***: concatenation of different features altogether) helps the performance of machine learning, but *ICSD* added the knowledge represented as HIN significantly outperforms other baselines. This again demonstrates that, to detect the insecure code snippets, *ICSD* utilizing both code content and social coding properties represented by the HIN is able to build the higher-level semantic and structural connection between code snippets with a more expressive and comprehensive view and thus achieves better detection performance.

Table 4: Comparisons of other machine learning methods

Metric	NB			SVM			*ICSD*
	f-1	***f-2***	***f-3***	***f-1***	***f-2***	***f-3***	
ACC	0.7757	0.6597	0.8161	0.8064	0.6904	0.8494	0.9118
F1	0.8002	0.6914	0.8372	0.8278	0.7208	0.8675	0.9190

4.5 Evaluation of Parameter Sensitivity, Scalability, and Stability

In this set of experiments, based on the dataset described in Section 4.1, we first conduct the **sensitivity** analysis of how different choices of parameters (i.e., walks per node r, walk length l, vector dimension d, and neighborhood size w) will affect the performance of *ICSD* in insecure code snippet detection. From the results shown in Figure 7(a) and 7(b), we can observe that the balance between computational cost (number of walks per node r and walk length l in x-axis) and efficacy (F1 in y-axis) can be achieved when $r = 10$ and $l = 60$ for insecure code snippet detection. We also examine how vector dimension (d) and neighborhood size (w) affect the performance. As shown in Figure 7(c), we can see that the performance tends to be stable once d reaches around 300; similarly, from Figure 7(d) we can find that the performance inclines to be stable when w increases to around 8. Overall, *ICSD* is not strictly sensitive to these parameters, and is able to reach high performance under a cost-effective parameter choice.

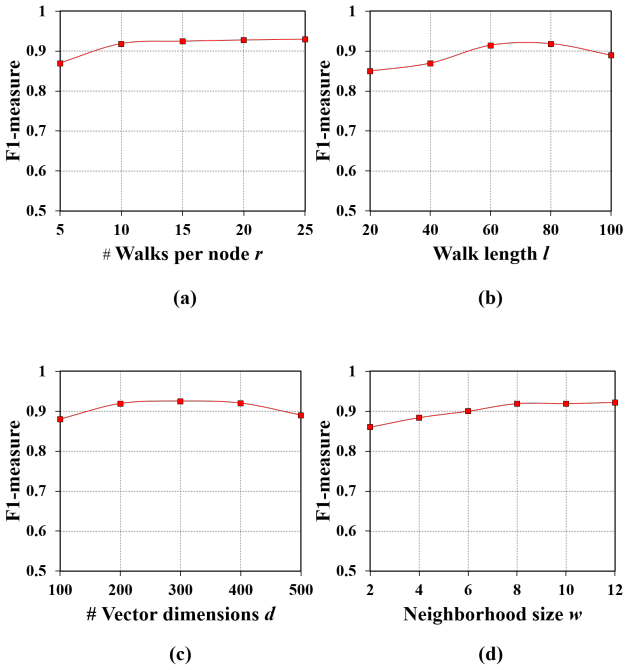

Figure 7: Parameter sensitivity evaluation.

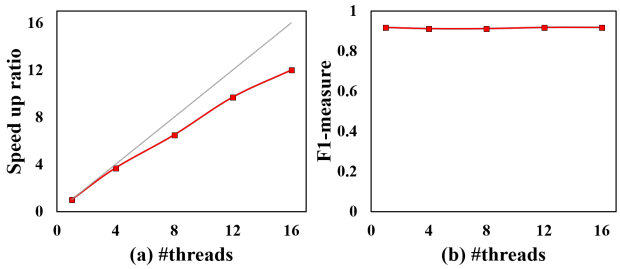

Figure 8: Scalability evaluation.

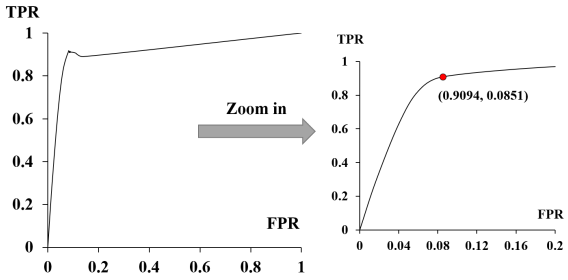

Figure 9: Stability evaluation.

We then further evaluate the **scalability** of *ICSD* which can be parallelized for optimization. We run the experiments using the default parameters with different number of threads (i.e., 1, 4, 8, 12, 16), each of which utilizes one CPU core. Figure 8(a) shows the speed-up of *ICSD* deploying multiple threads over the single-threaded case, which reveals that the model achieves acceptable sub-linear speed-ups as the line is close to the optimal line; while Figure 8(b) shows that the performance remains stable when using multiple threads for model updating. Overall, the proposed system are efficient and scalable for large-scale HIN with large numbers of nodes. For **stability** evaluation, Figure 9 shows the receiver operating characteristic (ROC) curves of *ICSD* based on the 10-fold cross validations; it achieves an average 0.9094 TP rate (*TPR*) at the 0.0851 FP rate (*FPR*) for insecure code snippet detection.

4.6 Case Studies

To better understand and gain deeper insights into the security-related risks of modern social coding platform of Stack Overflow, in this section, based on our developed system *ICSD*, we further analyze the detected insecure code snippets in Stack Overflow. Table 5 shows different types of security risks or vulnerabilities that could result from the detected insecure code snippets.

From Table 5, we can observe that the most prevalent insecure code infiltration for Android apps in Stack Overflow is Android Manifest configuration (28.73%), which would pose serious threats to Android apps, since Manifest retains all the components, security mechanisms, and structure information for an app [8, 9]. Such detected insecure code snippets related to Android Manifest configuration vulnerabilities include violation of least permission request, the component features being configured as exported, and data backup and debuggable setting being turned on, etc. For example, as shown in Figure 10.(a), many unnecessary permissions are requested in the detected insecure code snippet, which could be exploited by cyberattackers to perform the attacks on Android apps. Actually, this code snippet was provided by an inexperienced user answering a Facebook problematic login question; but it was also copied-pasted by other users in their answer threads responding to different posted questions. From Table 5, we can also observe that data security is another kind of prevalent insecure code infiltration (23.05%). After further analysis, the vulnerabilities of data security mainly focus on plaintext transmission, shared preferences, open file outputs, and external storage being set to readable/writable. The example of such kind of insecure code snippet is shown in Figure 10.(b), which uses cleartext username and password for FTP authentication instead of Secure File Transfer Protocol (SFTP), where password sniffing attacks could be performed to collect username and password that would cause sensitive information leakage.

Table 5: Types of security risks of detected insecure codes

Types of security risks	# Detected Codes	Percentage
Android Manifest configuration	2,601	28.73%
WebView component	271	02.99%
Data security	2,087	23.05%
File directory traversal	1,413	15.60%
Implicit intents	851	09.40%
Security checking	1,831	20.22%

```
<uses-permission
 android:name="android.permission.READ_CONTACTS"/>
<uses-permission
 android:name="android.permission.CALL_PHONE"/>
<uses-permission
 android:name="android.permission.SEND_SMS"/>
<uses-permission
 android:name="android.permission.WRITE_STORAGE"/>
<uses-permission
 android:name="android.permission.INTERNET "/>
<uses-permission
 android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
 android:name="android.permission.ACCESS_WIFI_STATE"/>
```

(a)

```
File file = new
 File(Environment.getExternalStorageDirectory()+"/pdf");
file.mkdir();
Try {
  ftp = new FTPClient();
  ftp.connect("ftpURL",21);
  int reply = ftp.getReplyCode();
  if (FTPReply.isPositiveCompletion(reply)) {
  ftp.login("username","password");
  ftp.enterLocalPassiveMode();
  transferFile(ftp); }
} catch(Exception e) {
  handleThrowable(e);
}
```

(b)

Figure 10: Insecure code snippets related to (a) Android Manifest configuration and (b) FTP data security.

The study based on the detected insecure code snippets in Stack Overflow using our developed system *ICSD* demonstrates that knowledge gained from social coding platform data mining can facilitate the understanding and thus help enhance its code security in modern software programming ecosystem.

5 RELATED WORK

There have been many works on knowledge discovery from Stack Overflow data [2, 5, 7, 11, 26–29, 44, 45] - from gamification motivation for voluntary contributions [7], discussion interest trend [27, 28], patterns of questions/answers [44] and project-specific language differences [26], to developer interaction [2], dynamics of the participation [5], repair patterns from extracted code samples [29] and interplay between platform activities and development process [45]. However, most of these works have focused in Stack Overflow semantics and user's behavior but rarely addressed the issue of code security analysis. The only exceptions appear to be [1] and [17] which both exploited Android app codes as a case study to evaluate the security of information source in Stack Overflow. Though those research results are promising, [1] only performed empirical studies while [17] merely analyzed the code snippet itself without considering any relationship to other Stack Overflow data (i.e., without utilizing the social coding properties in this platform). Different from the existing works, in this paper, to detect the insecure code snippets in Stack Overflow, we propose to utilize not only the code content, but also various kinds of relationships among users, badges, questions, answers, and code snippets. Based on the extracted relation features, the code snippets are depicted by a structured HIN.

HIN is used to model different types of entities and relations [37], which has been intensively deployed to various applications, such as scientific publication network analysis [39, 41], social network analysis [15, 16], and malware detection [14, 23]. To reduce the high computation and space cost in network mining, many efficient network embedding methods have been proposed, including homogeneous network representationn learning (e.g., DeepWalk [34], node2vec [19], PTE [42], and LINE [43]) and HIN representation learning (e.g., ESim [36], metapath2vec [13] and HIN2vec [18]). Unfortunately, these methods cannot be directly employed in our application, which is to exploit social coding properties in addition to code content for automatic detection of insecure code snippets. To tackle this challenge, in this paper, we propose a novel learning model named *snippet2vec* for node (i.e., code snippet) representation learning in HIN where both the HIN structures and semantics are maximally preserved; after that, a multi-view fusion classifier is constructed for insecure code snippet detection.

6 CONCLUSION

To address the imminent code security issue in modern social coding platforms, in this paper, we bring an important new insight to exploit social coding properties in addition to code content for automatic detection of insecure code snippets in Stack Overflow. To depict the code snippets, we not only analyze the code content, but also utilize various kinds of relations among users, badges, questions, answers, code snippets and keywords in Stack Overflow. To model the rich semantic relationships, we first introduce a structured HIN for representation and then use meta-path based approach to incorporate higher-level semantics to build up relatedness over code snippets. Later, we propose a novel network embedding model named *snippet2vec* for representation learning in the HIN where both the HIN structures and semantics are maximally preserved. After that, a multi-view fusion classifier is built for insecure code snippet detection. The experimental results based on the data collections from Stack Overflow demonstrate that the developed system *ICSD* integrating our proposed method outperforms alternative approaches in insecure code snippet detection. The proposed method and developed system can also be easily expanded to code security analysis in other social coding platforms, such as GitHub and Stack Exchange.

ACKNOWLEDGEMENT

The authors would also like to thank the anti-malware experts of Tencent Security Lab for the helpful discussion and data annotation. This work is partially supported by the U.S. National Science Foundation under grants CNS-1618629, CNS-1814825 and OAC-1839909, NIJ 2018-75-CX-0032, WV Higher Education Policy Commission Grant (HEPC.dsr.18.5), and WVU Research and Scholarship Advancement Grant (R-844).

REFERENCES

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You Get Where You're Looking For The Impact of Information Sources on Code Security. In *IEEE Symposium on Security and Privacy (SP)*. 289–305.

[2] Tanveer Ahmed and Abhishek Srivastava. 2017. Understanding and evaluating the behavior of technical users. A study of developer interaction at StackOverflow. *Hum. Cent. Comput. Inf. Sci.* 7, 8 (2017).

[3] AttackFlow. 2017. Watch Out For Insecure StackOverflow Answers. In *https://www.attackflow.com/Blog/StackOverflow*.

[4] Léon Bottou. 1991. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes* 91, EC2 (1991).

[5] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2018. How to ask for technical help? Evidence-based guidelines for writing questions on Stack Overflow. *Information and Software Technology* 94 (2018), 186–207.

[6] Andrea Capiluppi, Alexander Serebrenik, and Leif Singer. 2013. Assessing technical candidates on the social web. In *IEEE Software*. 45–51.

[7] Huseyin Cavusoglu, Zhuolun Li, and Ke-Wei Huang. 2015. Can Gamification Motivate Voluntary Contributions? The Case of StackOverflow Q&A Community. In *Proceedings of the 18th ACM conference companion on computer supported cooperative work & social computing*. 171–174.

[8] Lingwei Chen, Shifu Hou, and Yanfang Ye. 2017. SecureDroid: Enhancing Security of Machine Learning-based Detection against Adversarial Android Malware Attacks. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC)*. 362–372.

[9] Lingwei Chen, Shifu Hou, Yanfang Ye, and Shouhuai Xu. 2018. DroidEye: Fortifying Security of Learning-based Classifier against Adversarial Android Malware Attacks. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*.

[10] John Coogle, Jeet Gajjar, and Chase Greco. 2017. StackInTheFlow: StackOverflow Search Engine. In *VCU Capstone Design Expo Posters*.

[11] Daniel Czyczyn-Egird and Rafal Wojszczyk. 2016. Determining the Popularity of Design Patterns Used by Programmers Based on the Analysis of Questions and Answers on Stackoverflow.com Social Network. In *Communications in Computer and Information Science (CCIS)*. 421–433.

[12] S. Deterding. 2012. Gamification: designing for motivation. *Interactions* 19, 4 (2012), 14–17.

[13] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'2017)*. 135–144.

[14] Yujie Fan, Shifu Hou, Yiming Zhang, Yanfang Ye, and Melih Abdulhayoglu. 2018. Gotcha-Sly Malware! Scorpion: A Metagraph2vec Based Malware Detection System. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. ACM, 253–262.

[15] Yujie Fan, Yiming Zhang, Yanfang Ye, and Xin Li. 2018. Automatic Opioid User Detection from Twitter: Transductive Ensemble Built on Different Meta-graph Based Similarities over Heterogeneous Information Network.. In *IJCAI*. 3357–3363.

[16] Yujie Fan, Yiming Zhang, Yanfang Ye, Xin Li, and Wanhong Zheng. 2017. Social Media for Opioid Addiction Epidemiology: Automatic Detection of Opioid Addicts from Twitter and Case Studies. In *CIKM*. ACM, 1259–1267.

[17] Felix Fischer, Konstantin Bottinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. 2017. Stack Overflow Considered Harmful? The Impact of Copy and Paste on Android Application Security. In *IEEE Symposium on Security and Privacy (SP)*. 121–136.

[18] Tao-Yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM)*. 1797–1806.

[19] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[20] Peter D Hoff, Adrian E Raftery, and Mark S Handcock. 2002. Latent space approaches to social network analysis. *J. Amer. Statist. Assoc.* 97, 460 (2002), 1090–1098.

[21] Shifu Hou, Aaron Saas, Lifei Chen, and Yanfang Ye. 2016. Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs. In *WIW '16*.

[22] Shifu Hou, Aaron Saas, Yanfang Ye, and Lifei Chen. 2016. DroidDelver: An Android Malware Detection System Using Deep Belief Network Based on API Call Blocks. In *International Conference on Web-Age Information Management (WAIM)*. 54–66.

[23] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. 2017. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'2017)*. ACM, 1507–1515.

[24] IDC. 2018. International Data Corporation (IDC). In *http://www.idc.com*.

[25] Ilse CF Ipsen and Carl D Meyer. 1995. The angle between complementary subspaces. *Amer. Math. Monthly* (1995), 904–911.

[26] David Kavaler, Sasha Sirovica, Vincent Hellendoorn, Raul Aranovich, and Vladimir Filkov. 2017. Perceived Language Complexity in GitHub Issue Discussions and Their Effect on Issue Resolution. In *ASE*. 72–83.

[27] Roy Ka-Wei Lee and David Lo. 2017. GitHub and Stack Overflow: Analyzing developer interests across multiple social collaborative platforms. In *International Conference on Social Informatics*. Springer, 245–256.

[28] Mario Linares-Vasquez, Gabriele Bavota, Massimiliano Di Penta, and Rocco Oliveto. 2014. How Do API Changes Trigger Stack Overflow Discussions? A Study on the Android SDK. In *ICPC*. 83–94.

[29] Xuliang Liu and Hao Zhong. 2018. Mining StackOverflow for Program Repair. In *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 118–129.

[30] Lucintel. 2017. Growth Opportunities in the Global Software Market. In *http://www.lucintel.com/software-market-2017.aspx*.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *arXiv preprint arXiv:1301.3781*.

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[33] Stack Overflow. 2018. Stack Overflow. In *https://stackoverflow.com/*.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*. 701–710.

[35] Sebastian Poeplau, Yanick Fratantonio, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. 2014. Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications. In *NDSS*. 23–26.

[36] Jingbo Shang, Meng Qu, Jialu Liu, Lance M. Kaplan, Jiawei Han, and Jian Peng. 2016. Meta-Path Guided Embedding for Similarity Search in Large-Scale Heterogeneous Information Networks. In *arXiv:1610.09769*.

[37] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2017. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2017), 17–37.

[38] StackExchange. 2018. StackExchange Statistics. In *https://stackexchange.com/sites#traffic*.

[39] Yizhou Sun, Rick Barber, Manish Gupta, Charu C Aggarwal, and Jiawei Han. 2011. Co-author relationship prediction in heterogeneous bibliographic networks. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 121–128.

[40] Yizhou Sun and Jiawei Han. 2012. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery (SLDMKD)* 3, 2 (2012), 1–159.

[41] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment (PVLDB)* 4, 11 (2011), 992–1003.

[42] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1165–1174.

[43] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW '15 Proceedings of the 24th International Conference on World Wide Web*. 1067–1077.

[44] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. 2011. How do programmers ask and answer questions on the web?: Nier track. In *33rd International Conference on Software Engineering (ICSE)*. 804–807.

[45] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. StackOverflow and GitHub: Associations Between Software Development and Crowdsourced Knowledge. In *International Conference on Social Computing (SocialCom)*. 188–195.

[46] Shuicheng Yan, Dong Xu, Benyu Zhang, Hong-Jiang Zhang, Qiang Yang, and Stephen Lin. 2007. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)* 29, 1 (2007), 40–51.

[47] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. 2017. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)* 50, 3 (2017), 41.

[48] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'2017)*. 635–644.

[49] Guido Zuccon, Leif A Azzopardi, and CJ Van Rijsbergen. 2009. Semantic spaces: Measuring the distance between different subspaces. In *International Symposium on Quantum Interaction*. Springer, 225–236.