Hindawi
International Journal of Intelligent Systems
Volume 2023, Article ID 4464974, 22 pages
https://doi.org/10.1155/2023/4464974

WILEY | Hindawi

Research Article

SecTKG: A Knowledge Graph for Open-Source Security Tools

Siqi Sun,[1] **Cheng Huang**,[1,2] **Tiejun Wu**,[3] **and Yi Shen**[2]

[1]*School of Cyber Science and Engineering, Sichuan University, Chengdu 610065, China*
[2]*Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China*
[3]*NSFOCUS Technologies Group Co., Ltd., Beijing 10089, China*

Correspondence should be addressed to Cheng Huang; opcodesec@gmail.com

Received 2 January 2023; Revised 27 July 2023; Accepted 28 July 2023; Published 14 August 2023

Academic Editor: Said El Kafhali

As the complexity of cyberattacks continues to increase, multistage combination attacks have become the primary method of attack. Attackers plan and organize a series of attack steps, using various attack tools to achieve specific goals. Extracting knowledge about these tools is of great significance for both defense and tracing of attacks. We have noticed that there is a wealth of security tool-related knowledge within the open-source community, but research in this area is limited. It is challenging to achieve large-scale automated security tool information extraction. To address this, we propose automated knowledge graph construction architecture, named SecTKG, for open-source security tools. Our approach involves designing a security tool ontology model to describe tools, users, and relationships, which guides the extraction of security tool knowledge. In addition, we develop advanced entity recognition and classification methods, ensuring efficient and accurate knowledge extraction. As far as we know, this work is the first to construct the large-scale security tool knowledge graph, containing 4 million entities and 10 million relationships. Furthermore, we investigate the tendencies and particularities of security tools based on the SecTKG and developed a security tool influence-measuring application. The research fills a gap in the field of automated security tools' knowledge extraction and provides a foundation for future research and practical applications.

1. Introduction

As modern cyberattack techniques continue to develop, multistage combination attacks have become increasingly prevalent, which presents significant challenges in attack detection and tracing. The State of Security 2022 report reveals that 64% of respondents reported difficulty in ensuring security, representing a 49% increase from the previous year [1]. The expansion of the attack surface, the complexity of the cybersecurity stack, and the shortage of security technicians are the most common reasons cited. The execution of multistage combination attacks involves careful planning and coordination by the attackers to carry out a series of attack steps. These steps are aimed at achieving specific goals at different stages via using a variety of attack tools and techniques. Therefore, automating the identification and analysis of these tools, as well as grasping their usage characteristics, is crucial for security researchers.

Research studies show that a significant proportion of hackers or security developers utilize open-source tools or release the tools they developed to the open-source platforms [2]. And the number of these tools and developers is growing rapidly [3]. Therefore, identifying and analyzing the information of these open-source security tools as well as the developers associated with these security tools can provide valuable insights for detecting and tracking cyberattacks. Studies [4, 5] surveyed security developers and security tools in GitHub, followed by manual analysis to identify potential conclusions that could help security professionals, while these two works are manual, which cannot be extended to large-scale data analysis. Studies [6–8] identified vulnerabilities and malware source codes in GitHub using automated or semiautomated methods. However, these studies are restricted to recognize specific categories, without taking into account the technical details and relationships of tools. Moreover, some open-source projects collect and categorize open-source security tools to help security professionals.

AlphaSeclab cataloged over ten thousand security-related tools based on GitHub repository star ranking [9]. Kali boasted a more comprehensive list of over 600 tools divided into 12 categories [10]. Yet, the timeliness of these security tool lists is largely insufficient. Only a few tools will be updated after 3-4 months even for the well-maintained Kali tools, and the tools' list maintained by individuals may remain unchanged for a year or even longer. It is passive and untimely for security professionals to obtain security tools' information from the abovementioned sources.

Due to the freely developed nature of most GitHub projects, extracting information from the vast amount of unstructured texts becomes challenging, and the security domain's complex semantics and specialized vocabularies make general domain methods unsuitable for direct application. While most of the research in security domain [11–14] mainly relies on CTI reports for security intelligence extraction, their application to open-source tools' data yields unsatisfactory results. It is important to design appropriate methods which consider the characteristics of open-source tools to ensure the quality of knowledge extraction for security tools. Meanwhile, research needs to be done on constructing a security tools' ontology that facilitates understanding and reuse of the knowledge. Although various cybersecurity ontology models [14–18] have been proposed, which are not applicable to security tools, existing security ontologies mostly focus on the description of the attack events, yet largely overlook the attack tools used.

In this work, we study how to automatically evaluate the security tools in the open-source community, including tools' application scenarios, technical characteristics, popularity, and related security users. Furthermore, build a large open-source security tools' knowledge base for security professionals. On the one hand, this work can serve as a guide and reference for penetration testers, enabling them to quickly grasp the applicable scenarios and technical characteristics of various tools. On the other hand, the portrayal of tools and associated users can assist in attack detection and traceability. By associating security tools' features, vulnerabilities, consequences, and relevant developers, defenders can trace the source of the fragmented attack information to the associated security tools and people. Our study combines knowledge graph technology with security tools and proposes a knowledge graph of security tools called SecTKG. Specifically, we design a security tools' ontology model that characterizes the attributes of security tools, security developers, and their relationships, which serves as the foundation of the knowledge graph. We develop page-level and paragraph-level entity classification models as well as a named entity recognition model based on the BiLSTM-Attention algorithm in order to achieve efficient and automated knowledge extraction. In the end, we explore the tendency of the security tools and implement a security tools' influence-measuring application based on the constructed SecTKG.

In general, we have made the following contributions:

(i) We design a security tools' ontology model which characterizes the properties of these tools, their users, and their associations in the open-source community. The ontology can provide a comprehensive and general reference for portraying open-source security tools.

(ii) To address the difficulty of knowledge extraction caused by the nonstandard, semantic complexity and highly specialized characteristics of open-source security tools' data, we implement different entity recognition methods according to the features of different entity types to realize efficient and automatic knowledge extraction. Each of proposed models achieves up to 85% precision and 82% recall.

(iii) We build the first large-scale knowledge graph for open-source security tools called SecTKG. 15,778 security tools are identified; a total of 4 million entities and 10 million relationships are incorporated in SecTKG. The code for the key components is available at https://github.com/das-lab/SecTKG.

(iv) We investigate the tendency and particularity of security tools and implement a security tools' influence-measuring application based on the SecTKG. The result shows that the application can identify some uncommon security tools for different attack phases.

The remainder of this paper is organized as follows: Section 2 presents related work in the area of open-source community and cybersecurity knowledge graphs. In Section 3, we introduce the ontology. Section 4 details the extraction approach implementation and Section 5 conducts experiments. The results and discussion are provided in Section 6. We explain the limitations of our work in Section 7 and conclude with Section 8.

2. Related Work

We review related works in two key areas: studies on the open-source community and studies on cybersecurity knowledge graphs.

2.1. Open-Source Community. As mentioned before, the vigorous development of the open-source community has greatly changed the development and use of software tools. Researchers want to find the trend and impact of community development and have also published many related studies [19].

Many studies focused on repositories' code or commit information and attempted to find code vulnerabilities. Perl et al. [6] conducted the first large-scale mapping of CVEs to GitHub vulnerable commits and proposed a new method to find potential vulnerabilities. Unruh et al. [20] found that the vulnerable code snippets contained in top-ranked tutorials can be used repeatedly. The analysis framework they proposed has found a total of 117 vulnerabilities by analyzing 64,415 PHP repositories on GitHub. Meli et al. [7] conducted the first large-scale

analysis of secret leakage on GitHub. By examining billions of GitHub files, they found that secret leakage affected over 100,000 repositories, which may place developers at a persistent risk. Rokon et al. [8] focused on the limitation of the lack of malware source code. A total of 7504 malware repositories were systematically identified from a large number of public archives for the first time. They also identified and profiled some professional hackers. Later, based on this work, Islam et al. [3] analyzed the hacker ecosystem by modeling the relationship between users and malware repositories. Qian et al. [21] developed an adversarial heterogeneous graph model, capturing relationships between repositories, developers, code, and unlabeled data, and utilized contrastive learning techniques to detect malicious repositories.

A number of studies focused on relevant repositories or developers' recommendation tasks. Acar et al. [4] studied the security programming ability of users in the open-source community by recruiting them to complete tasks, demonstrating potential for more efficient recruitment. They also investigated how to better support open-source projects in terms of trust and security considerations in their subsequent work [22]. Many efforts modeled the interaction nature of the open-source community to identify influential repositories or users. Zhang et al. [23] combined the activities of users in the open-source community to recommend suitable developers for the project in a wider range. Wan et al. [24] introduced a probability model to evaluate the professional knowledge of developers in the open-source community and help search for experts. Zhou et al. [25] modeled the complex relationships in the open-source community into a heterogeneous information network to support further analysis.

Some studies categorized the tools to support researchers in continuing a comprehensive and structured survey. Hoque et al. [26] systematically categorized tools into two classes, defender tools and attacker tools, and detailed the subclasses of each. The popular security tools of the public domain are introduced and analyzed in detail by category for a better understanding of their capabilities. Kaksonen et al. [5] related Google search, tweets, and SecTools.org with GitHub stars to estimate the popularity of open-source InfoSec tools. Furthermore, they analyzed the top 100 tools used for cybersecurity, where information would be useful in relevant security research. Sas and Capiluppi [27] evaluated different software classification schemes in existing studies, summarized the common seven types of problems, discussed how to improve software classification, and facilitated the research.

Overall, these studies suggest that it is necessary and meaningful to mine security-related information in the open-source community. Nevertheless, the abovementioned research did not propose a comprehensive automatic knowledge extraction method for open-source security tools. Moreover, most of the research focused on classifying the tools and omitting tools' technical details. In this paper, we take the first step in this direction by building a knowledge graph for security tools in the open-source community.

2.2. Cybersecurity Knowledge Graph. Google formally proposed knowledge graph in 2012, and it has received a lot of attention since then [28]. Knowledge graph technology can help computers understand the logical relationship between words, concepts, and elements. Peng et al. [29] conducted a survey on knowledge graphs, indicating that knowledge graphs can significantly enhance the performance of AI systems and hold promising prospects for driving innovation and advancement across multiple areas. Combining knowledge graph technology with the field of cybersecurity, building security domain knowledge graphs can organize the heterogeneous and fragmented security information in the network into a structured and interrelated knowledge base and provide support for threat discovery, attack traceability, and expert decision-making.

The knowledge graph construction process usually includes ontology construction, entity recognition, and relationship extraction. Ontology is a way of showing the properties of a subject area and how they are related by defining a set of concepts and categories that represent the subject [30]. The ontology is the foundation of the entire knowledge graph system and provides guidance on entity recognition and relationship extraction. The first step in building a cybersecurity knowledge graph is to abstract massive security domain data into higher-level concepts and build a corresponding ontological model. Owning cybersecurity data has the characteristics of polysemy and specialization; the data characteristics, domain knowledge, and expert experience should be taken into consideration to build the security ontological model. A number of research studies have been established to create ontological models focused on security-related concepts and have produced significant previous work.

Undercoffer et al. [15] proposed the first published research that formally defined IDS ontology for use in intrusion detection. In the IDS ontology, the host is defined as the top-level class, and the attack, system component, consequence, input, and means classes are defined as second level. Corresponding attributes and subclasses are defined for each class. At last, the Mitnick attack is successfully inferred based on the constructed ontology in the example. Iannacone et al. [16] also proposed a cybersecurity ontology called STUCCO. They were concerned about the challenges in managing and utilizing the increasing amount of cybersecurity information. By constructing STUCCO, the multisource data were integrated into unified and organized concepts. The research made contributions to the convenient and reusable presentation of security information. Syed et al. [17] defined a unified cybersecurity ontology (UCO) based on the IDS ontology. The UCO is in accord with STIX standard and also extended with CVE, CVSS, Cyber Kill Chain, and STUCCO. UCO helps support information integration and cyber situational awareness in cybersecurity systems. Pingle et al. [18] created UCO 2.0 based on UCO and STIX 2.0 standard. The RelExt system was implemented to predict relationships defined in UCO 2.0.

Cyber Kill Chain (CKC) oriented and derived attack models have proved to be popular and effective. The CKC model describes how attackers conduct attacks and achieve

their objectives and divides the attack process into 7 steps [31]. By understanding the different stages of the attack process, organizations can use various techniques and tools to prevent or detect attacks and improve their security defense capabilities. The ATT&CK model was built on the basis of the CKC and focused on describing the attack tactics and techniques used by attackers during the attack process [32]. ATT&CK describes the adversary's actions and specific defense measures from tactics, techniques, software, mitigation, and adversary group five aspects, providing organizations and security professionals with a detailed attack knowledge base. However, the ATT&CK model cannot help engineers address those threats from an engineering perspective very well. Therefore, the D3FEND model was created to map cybersecurity countermeasures to offensive TTPs later [33]. D3FEND defines both the key concepts in the cybersecurity countermeasure domain and the relations necessary to link those concepts to each other.

On the basis of previous work, researchers studied the entity recognition and relation extraction method in the cybersecurity corpus to improve the quality of knowledge in the cybersecurity knowledge graph. Jia et al. [34] proposed a quintuple cybersecurity knowledge graph construct and deduct approach, which considered the aspects of the concept, instance, relation, properties, and rule. Guo et al. [11] improved the effect of entity recognition and relation extraction in UCO 2.0 and proposed a joint entity and relation extraction model called CyberRel. Sarhan and Spruit [12] first applied the open information extraction method in the cybersecurity knowledge graph construct. They proposed a CTI knowledge graph called Open-CyKG, which aims to overcome the limitations of a prespecified set of information. Li et al. [13] proposed an improved graph alignment algorithm for matching attack graphs created from MITRE ATT&CK techniques and extracted from CTI reports and formed a technique knowledge graph called AttacKG. Ren et al. [14] constructed a security knowledge graph for APT organization attribution by recognizing threat knowledge from bilingual CTI reports combined with STIX and CYBOX standards. Kaiser et al. [35] proposed a multisource-fused threat knowledge base and utilized graph analysis to generate attack hypotheses to aid security analysts. Some studies extracted entities and relationships from security-related papers to aid researchers in more efficient literature searches. Dessí et al. [36] constructed a large-scale knowledge graph called CS-KG by extracting knowledge from computer science-related papers using publicly available tools.

In summary, existing security ontological models and knowledge graphs overlooked the characterization of security tools to a certain extent. These ontological models primarily focus on the characterization of attack events, with little emphasis on the features and functionalities of attack tools. Although the ATT&CK and D3FEND knowledge bases describe the technical and functional aspects of some tools, the quantity of tools is limited and primarily reliant on security reports. Considering that knowledge graphs are constructed by extracting entities defined in ontological models, the research on existing knowledge graphs has also overlooked the extraction of tool-related knowledge.

3. Security Tools' Ontology

The ontological model serves as the foundation for the SecTKG, aiding in the better organization and expression of knowledge and facilitating knowledge sharing. The objective of the ontological construction is to encapsulate knowledge of security tools in the open-source community with more professional and unified terminologies, depicting their application scenarios, technical characteristics, popularity, and relevant security users. A total of 9 entity types and 7 relationships are defined in the ontology by following the seven-step method. Figure 1 presents an overview of the security tools' ontology.

3.1. Ontological Design. After the research motivation is clarified, the ontological model is designed based on the seven-step method [37] with the consideration of data characteristics, domain knowledge, and expert experience, and the effectiveness of the ontology is evaluated. For the open-source community, some entities and relationships are existing and obvious, such as users, repositories, and their relations. However, these features are superficial and lack outer join, which is not enough to meet the research motivation. Accordingly, deep features need to be abstracted from the available information to better assist penetration testers and security researchers. Defining the entity types and relationships is challenging but critical for achieving our research objectives.

Here, we explain the key parts of the security tools ontological construction.

3.1.1. Preliminary Work. First of all, we investigated the existing ontology research in the field of cybersecurity, but not limited to the research listed in the related work. We reviewed the classes and relationships defined in existing cybersecurity ontologies, preserving those that could be applied to the security tool ontology. However, these ontologies lack the characterization of tool features. Inspired by the attack models, we reshaped the global process to describe the security tool's applicable scenarios from the perspective of attacks. CKC and ATT&CK are currently the most widely used attack models. CKC outlines how attackers execute attacks in seven steps: reconnaissance, weaponization, delivery, exploitation, installation, command and control, and action. ATT&CK categorizes attackers' methods of executing attacks into 14 tactics and hundreds of techniques. In summary, CKC focuses on describing a complete attack process, while ATT&CK focuses on the attack techniques used by attackers during the attack process. We design a security tool attack phase classifier based on the definitions of attack steps in the CKC and attack tactics and techniques in the ATT&CK.

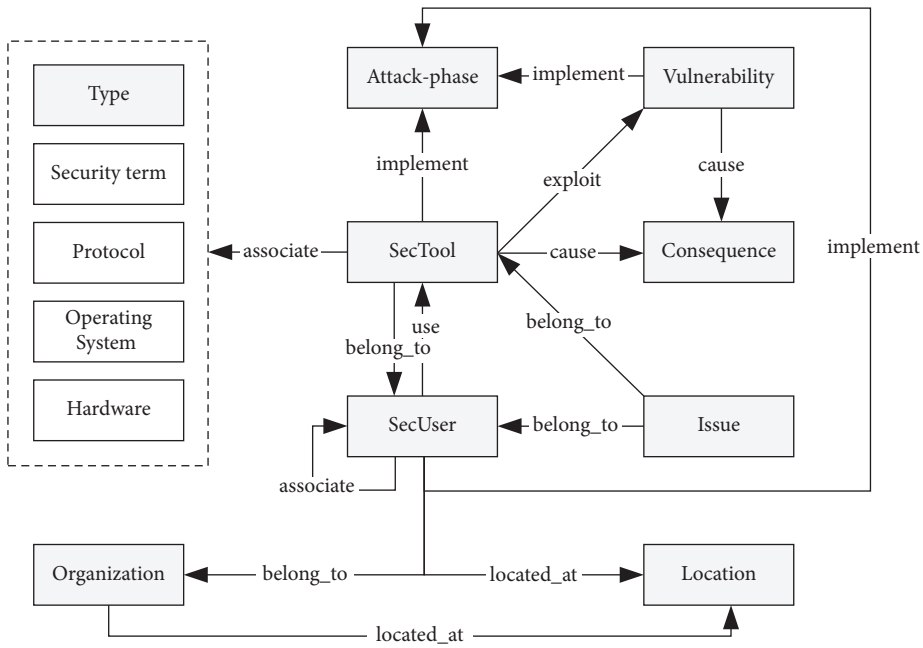

Figure 1: Security tools' ontology.

3.1.2. Definition of Important Classes and Relationships. In ontological construction, we investigated the current popular open-source community and thousands of actual open-source tools' data. The open-source tools' data collection work is detailed in Section 5.1. All the elements are listed to summarize the commonalities and then refined to more general classes and relationships that are not restricted to a particular open-source community.

3.1.3. Ontological Evaluation. Three domain experts are invited to help evaluate and iterate the security tools ontology. It is evaluated from three aspects of class nonintersection, generality, and integrity. Class nonintersection standard is to ensure that any two classes of entities do not intersect, and there is no communal instance between classes. Generality standard evaluates the adaptability of the ontology model and its ability to meet different application requirements, and whether it can be associated and compatible with the existing model. Integrity standard ensures that the ontological model can completely describe the characteristics and connections of the open-source community security tools. Finally, after four rounds of modification and iteration, the ontological model defines a total of 9 entity types and 7 relations.

3.2. Entities in Security Tools' Ontology. Definition of entity types affects the subsequent knowledge extraction to a great extent. Incomplete entity type definitions lead to the omission of security tool knowledge, and unclear entity type definitions may impact the effect of knowledge extraction. Our entity types are defined based on security tools, and they also cover key features of the tools as well as the relevant security developers and organizations. The details of these entity types are introduced as follows:

(i) SecTool: This entity refers to an open-source repository that could be used to launch attacks. It can be encapsulated software or just a script. This entity has the following attributes: programming language, readme file, description, topics, home page, security term, protocol, operating system, hardware, created time, and updated time.

(ii) SecUser: This entity refers to an open-source platform user who is concerned about or has developed open-source security repositories. This entity has the following attributes: user name, name, blog, e-mail, Twitter name, company, public repositories number, created time, and updated time. This entity is designed to create a security-related programmers' database.

(iii) Attack-phase: This entity refers to an attack stage that describes the specific role of the repositories in the overall attack process. It is limited to 5 phases: reconnaissance, exploit, persistence, lateral movement, and actions. Attack-phase entity maps an open-source repository into a particular phase of an attack life-cycle. Table 1 shows the definition of these 5 attack phases.

(iv) Vulnerability: This entity refers to a patch of bug or weakness of systems or software that could be exploited by hackers. If an open-source repository implements an attack by exploiting vulnerabilities, it can extract relevant vulnerability names or links from it.

(v) Type: This entity refers to an open-source repository category to describe the repository as a tool that works on the system level or the level above. The definition of system tool and

Table 1: Attack-phase definition.

Category	Definition
Reconnaissance	A phase aims to collect information supporting future attacks, which can be used to perform initial access or determine the target range, e.g., scanning tools and network analysis tools
Exploit	A phase aims to discover and exploit vulnerabilities for further action, e.g., fuzzy testing tools, brute cracking tools, and injection tools
Persistence	A phase aims to avoid being discovered by the opponent in the whole operation, e.g., Trojan and backdoor tools
Lateral movement	A phase aims to obtain control authority and conduct intranet penetration in the opponent's environment, e.g., agent tools and network traffic attack tools
Actions	A phase aims to take actions to obtain or destroy the opponent's data information or affect the opponent's normal use of the system or network, e.g., sensitive information acquisition tools and resource hijacking tools

application tool is shown in Table 2. This entity type summarizes the unstructured description of tool types as a specified and structured category.

(vi) Consequence: This entity refers to an impact or result that will be caused on successfully applying the tool. It is limited to four categories: DenialOfService, LossOfConfiguration, LossOfInformation, and RemoteAccess. Table 3 shows the definition of four consequence categories. This entity standards the unstructured description text into the abovementioned four types of consequences.

(vii) Issue: This entity refers to a question of the tool's bug or feature request committed by developers who want to use this tool. This entity has the following attributes: issue title, issue body, issue status, and commit time.

(viii) Organization: This entity refers to a group or company where the security users belong. This entity type helps associate users and identify their belongings.

(ix) Location: This entity refers to a geographical position where the security user or organization is located. This entity portrays users and organizations in terms of geographical location.

For the abovementioned entities and entity-related attributes, different extraction methods are designed. SecUser, issue, organization, and location and the attributes related to these four entity types are collected by crawlers or matched by regular expressions. Attack-phase, consequence, and type entity types are extracted by the classification method, as they require understanding the context of the tools. Security tools' vulnerability, operating system, protocol, hardware, and related security terms are extracted by the named entity recognition method, since there is no unified naming law or writing rules for them. The detailed introduction of the relevant models is in Section 4.

3.3. Relationships in Security Tools' Ontology. Relationships connect entities in ontology to form a knowledge graph. Seven relationships are defined to associate entities defined in the security tools ontology, they are as follows:

(i) Use: A relationship indicating a SecUser entity is concerned about a SecTool entity. The subject and object of the relationship are SecUser entity and SecTool entity, respectively. This relationship connects the potential users of the SecTool.

(ii) belong_to: A relationship represents the object being subordinated to or proposed by the subject. This relationship indicates the assets of the SecUser entity or subordinate members of the organization entity.

(iii) Exploit: A relationship that the subject aims to make use of the object to attack victims. The subject is a SecTool entity and the object is a vulnerability entity.

(iv) Cause: A relationship represents the impact caused by the subject. The subject is a SecTool entity or vulnerability entity, and the object is a consequence entity.

(v) Implement: A relationship represents a SecTool or a SecUser performing an Attack-phase. This relationship connects SecTool and SecUser to the attack technique.

(vi) Associate: The relevant relationship between SecUser entities or between the SecTool entity and the SecTool entity's characteristics.

(vii) located_at: A relationship connects the location entity to the SecUser entity or organization entity.

Since the relationship set to be extracted has been determined in advance, the extractions are belonging to limit relationship extraction. The abovementioned relationships were extracted using a rule-based relationship extraction method, which was implemented in two cases. First, for use, belong_to, and located_at these three relationships and associate relationship between SecUsers, extract these relationships between entities by matching the corresponding fields from the crawling data. Second, for the rest relationships exploit, cause, implement, and associate relationship with SecTools, a set of rules are defined based on the security tools ontology to extract them. The relationship extraction method is elaborated in detail in Section 4.

Table 2: Type categories' definition.

Category	Definition
System tool	A tool that controls and coordinates computer and external equipment and supports application software development and operation
Application tool	A tool that works on the system tool and helps complete user requirements

Table 3: Consequence categories' definition.

Category	Definition
DenialOfService	Denial of service affects the service provided by the system
LossOfConfiguration	Leakage or destruction of the basic system and network information or permissions
LossOfInformation	Data leakage or destruction
RemoteAccess	Remote access and control, such as botnet, CC, and RAT

4. Knowledge Extraction Approach

Figure 2 shows the architecture of our SecTKG. The architecture can be divided into security tools ontology construction and SecTKG building two parts. The security tools' ontology is the foundation of the whole architecture. Four modules are designed for building the SecTKG. Firstly, the data acquisition module collects raw data from the open-source community by crawlers. Detailed information regarding the data acquisition methods and the quantity of data can be found in Section 5.1.1. Then, the data preprocessing module cleans the raw data and matches the fixed format knowledge by a regular expression method. The knowledge extraction module incorporates three distinct methods, each dedicated to recognize different types of entities. Finally, the application module generates the SecTKG, which can be used for knowledge fusion or inference.

The specific objective of this study was to effectively extract knowledge from the description texts of the open-source tools according to the ontological model. Accordingly, three methods are implemented to extract information according to the characteristics of different entities. For some entities or attributes that follow certain formats and rules, such as CVE vulnerability number, e-mail address, and IP address, they extract them directly through regular expression matching. The regular expression method is usually accurate, easy to implement, and able to reduce unnecessary resource consumption. For the entities that need to be understood in context or even the full text, classifiers based on the page-level and paragraph-level are trained for classification. For some entities whose names are not fixed such as operating system and hardware names, the named entity recognition (NER) method is adopted.

4.1. Data Preprocess. In the data preprocessing module, irrelevant characters will be deleted, the text format will be unified, and the knowledge that can be directly extracted by regular expression in the raw data will be extracted. First, we will delete all non-ASCII characters, delete the blank characters at the beginning and end of each line, process invalid and missing values, remove duplicate crawled data, and delete any duplicated information related to tools or users. Second, we will unify the text language and markdown format segmentation. Through the observation of the data, it is found that the project description of the open-source community mostly uses English. Considering the impact of the amount of data on the training effect, in the actual implementation process, the Chinese texts are translated into English texts with the help of the API provided by Google translation for unified processing. Also, most Readme files edit in markdown format and there are various symbols to display the title or subtitle (e.g., ## and multiple – or =). We conclude 6 markdown title expressions and replace them with the "@part" symbol. Finally, based on the regular expression, the entities or attributes with a fixed format will be extracted (e.g., e-mail, URL, code block, and file path).

4.2. Classification for Entity Classes' Extraction. For the entities that are not explicitly stated in the texts and need to be understood in context, extract them by the text classification method. Considering that the descriptions of open-source repositories are unstructured and diverse, it is one-sided to conclude the full text into statistical features. Therefore, the deep learning classification method is adopted to automatically learn features through neural networks.

Although the Readme file of the tool is unstructured and diverse, there are still some regular patterns in it. One general pattern is that most tools put a short introduction of the tool in the first paragraph to make it easier for others to read and use. Accordingly, the judgment of the tool's type or consequence can just rely on the first paragraph of the tool's Readme file. However, this paragraph of description is not enough to express all the information. The presence of the specific principles or vulnerabilities used by the tool may be distributed anywhere in the Readme text. In this paper, we implement page-level classification model and paragraph-level classification model. We manually annotate thousands of open-source tools for training, testing, and validation of each classifier. The specific categories and the corresponding number of labels are presented in Table 4.

4.2.1. Page-Level Classification for Attack-Phase Extraction of Security Tools. The attack phases of security tools are extracted through the page level classification method. In the

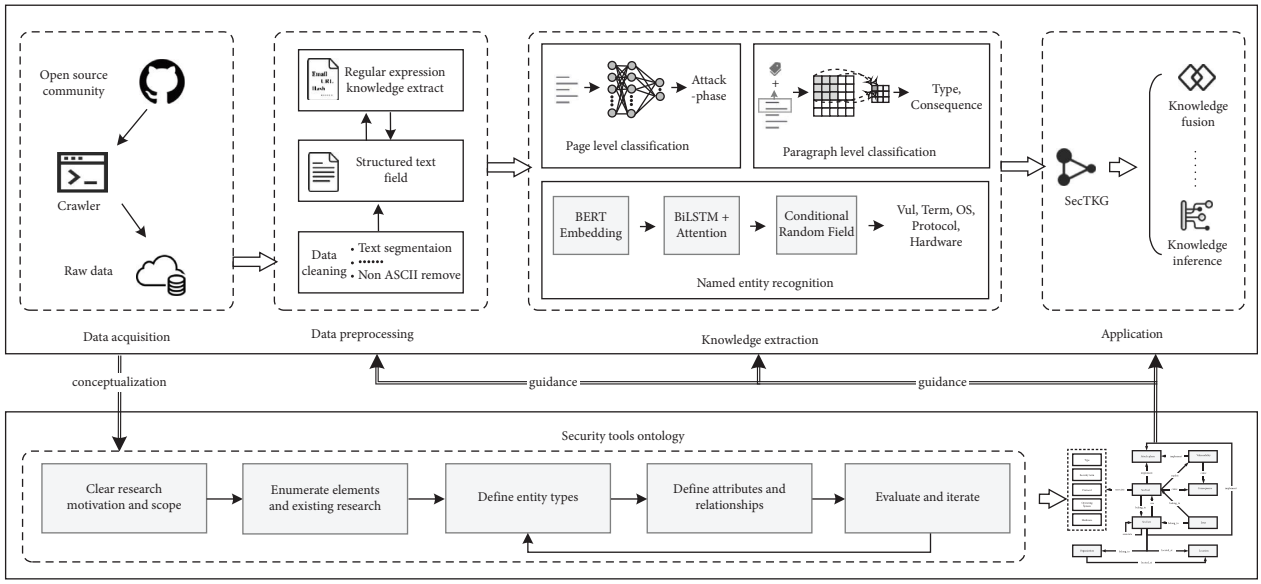

Figure 2: SecTKG architecture.

Table 4: Labeled dataset detail.

Entity name	Category	Number
Attack-phase	Reconnaissance	418
	Exploit	690
	Persistence	332
	Lateral movement	511
	Actions	187
	Others	692
Type	System tool	623
	Application tool	977
Consequence	DenialOfService	133
	LossOfConfiguration	360
	LossOfInformation	634
	RemoteAccess	164
	Others	400

open-source community, the textual features of a tool consisting of the following five parts: name, description, topics, Readme, and file and directory names. Among these, the Readme provides the most comprehensive and detailed explanation of the tool. After applying the data preprocessing module, the first 2000 words are extracted from the Readme text and transformed into word vector using a Word2Vec model. These word vectors then serve as input for the page-level classification model.

According to the statistics of the length of the Readme text, the character length of more than 44% of the data is more than 1000. Directly applying the convolutional neural network (CNN) or recursive neural network (RNN) to this long data sequence, the characteristics of the previous long time slice are easy to be covered after multistage calculation. This dependence issue can be settled by adopting long short-term memory (LSTM) [38]. Based on the original RNN hidden layer, the cell state and the gate units are added in LSTM hidden layer. For a long Readme text input sequence, it is necessary to discard irrelevant information in a timely manner. Whether to memorize or forget features will be controlled by the gate units. Each gate unit contains a sigmoid function σ to determine the probability of allowing information to pass. The output of σ is between 0 and 1. For time t, the input X_t is determined by the input gate i_t. The retention of cell state C_{t-1} is controlled by the forget gate f_t. The current output of the layer is mainly related to the previous training hidden state h_{t-1} and current input X_t. The activation function $\tan h$ maps the output to the range of −1–1. At last, the output gate o_t gives the value h_t as one of the unit outputs. The current output of the layer is mainly related to the hidden state h_{t-1} of the previous training and current input X_t. The derivation method is shown in equations (1)–(5). The weight matrix and bias vector are represented as W and b, respectively.

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, X_t\right] + b_f\right), \tag{1}$$

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, X_t\right] + b_i\right), \tag{2}$$

$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, X_t\right] + b_o\right), \tag{3}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tan h\left(W_C \cdot \left[h_{t-1}, X_t\right] + b_o\right), \tag{4}$$

$$h_t = o_t \cdot \tan h\left(C_t\right). \tag{5}$$

However, the LSTM can only calculate forward and cannot consider the impact of the current information on the previous information. Hence, page level classification model adopts the bidirectional LSTM (BiLSTM) [39], which consists of forward and backward LSTM networks. It can learn the characteristics of sequentiality and long-term dependencies and capture the dependencies implied in the Readme.

We consider that some keywords can strongly indicate the attack-phase of a security tool. For example, if the phrase "port scanner" is in the Readme text, it is likely a security tool

for reconnaissance phase. The attention mechanism is introduced to enable the model to focus on key information, much like humans do, and assign greater weight to important information in the input data. The attention mechanism function Attention(Q, K, V) maps query Q to key-value pairs $\{K: V\}$. By breaking the traditional encoder-decoder structure, the output of the encoder in each time step is combined with the output of the decoder in time step T, and a context vector is created for time step T in order to encapsulate the most relevant information in the encoder.

The page level classification model is based on BiLSTM and attention mechanism algorithm. First, the Readme text is transformed into word vectors using a Word2Vec model trained on security corpus. These word vectors are then fed as input sequences into the BiLSTM layer for feature extraction, capturing contextual information within the sequence. The extracted feature sequences = $\{h_1, h_2, h_3, \ldots, h_n\}$ are then passed to the attention layer, where the attention layer makes the model pay more attention to the important information in the text and reduce the interference from less relevant information on the output. In the attention layer, an attention matrix captures the similarity between each word and all adjacent words in the sentence and assigns weights to each word based on its importance. Finally, the output from the attention layer is transformed into a probability distribution for each class by the fully connected layer. The class with the highest probability is selected as the classification result.

4.2.2. Paragraph-Level Classification for Type and Consequence Extraction of Security Tools. The type and consequence of security tools are extracted based on the paragraph-level classification method. According to our observations, most open-source tools put a brief introduction to the tool in the first paragraph for the convenience of others to read and use. We only intercept the first paragraph of the Readme text as a paragraph-level classification feature and unite it with the tool's topics, attack-phase, and programming language features. Since most Readme files will use the standard markdown format, we segment the Readme text according to the paragraph symbol that is marked in the data preprocessing module. If a Readme file does not follow the markdown format, the text will be cut at the first 200 words.

The input text of the paragraph-level classification model intercepts the first paragraph in the Readme file and combines it with the topics, attack-phase, and programming language feature fields of the tool. The Word2Vec model trained on the security corpus is adopted to convert input text into sequence X_i and take X_i as the input of the paragraph-level classification model. Different from the input data of the page-level classification model, the input data X_i is short and the syntax logic is not obvious. Due to the characteristics of the input data of the paragraph level classification model, it is inappropriate to directly apply the page-level classification model to this task. This may cause unnecessary performance consumption and overfitting problems.

Considering the text features of paragraph-level classification described above, adopt the TextCNN proposed by Chen [40] to construct the model. The TextCNN's ability to process short text data and its simple network structure make it a suitable candidate to boost the performance of the paragraph-level classification model. For the input sequence X_i, the paragraph-level classification model first extracts features from X_i through convolutional layers. The filters in the convolutional layers learn different features in the input sequence and generate corresponding feature maps. Then, the pooling layer adopts the global max-pooling method to retain features and reduce the scale of parameters. Finally, the feature vectors are multiplied by the weight matrix of the model through fully connected layers, and the score for each class is obtained using an activation function. The class with the highest probability score serves as the prediction result of the paragraph-level classification model for the input sequence.

4.3. Named Entity Recognition for Irregular Attributes' Extraction. The task of named entity recognition in the field of cybersecurity usually involves identifying vulnerabilities, malwares, IP addresses, attack organizations, and other terms. For entities with a fixed format such as IP address, recognition can be carried out based on rule matching. For terms such as malware names that do not follow specific formats or syntax, the machine learning method is usually used to recognize them by automatically learning features through neural networks. The named entity recognition model in this work is used to recognize vulnerability entity type and four entity attributes associated with the SecTool entity, namely security term, network protocol, operating system, and hardware. For these five entity types, we annotate over 10,000 entities in the Readme texts of tools classified as security tools, and the specific number of labels for each entity type is provided in Table 5. The BIO annotation strategy is employed for the labeling process.

Although the named entity recognition technology for general fields has been relatively mature, recognizing security entities is still a challenging task. First, the writing method of cybersecurity entities is "nonstandard." On the one hand, entities often use a variety of types of combinations, such as word and number combinations (e.g., Windows 10) and noun and verb combinations (e.g., code injection). On the other hand, the abbreviations of entities are common and changeable. For example, "windows operating system" can be written as Win or win. Second, the semantics of cybersecurity entities are complex. The general phenomenon of polysemy can easily cause entity confusion. For example, the word "MAC" can refer to a MAC address, a Mac computer, or even a cosmetics brand. Third, the cybersecurity situation is complex and changeable, and new security entities are constantly emerging. Our method takes the characteristics and challenges of security texts into consideration to implement the named entity recognition method for security tools. This method is illustrated from two aspects: BERT-based word embedding and attention-based BiLSTM-CRF NER model.

Table 5: Labeled dataset distribution.

Model	Entity name	Number
Page-level classifier	Attack-phase	2,830
Paragraph-level classifier	Type	1,660
	Consequence	1,691
NER	Vulnerability	322
	Security term	3,278
	Protocol	3,387
	Operating system	3,239
	Hardware	1,077

4.3.1. BERT-Based Word Embedding. We used bidirectional encoder representation from transformers (BERT) [41] to construct a word vector. BERT uses a bidirectional transformer encoder to capture language features and dependencies between words from large-scale unlabeled data through huge internal training parameters, so as to achieve strong representation ability. In addition, a two-way transformer encoder solves the defect that traditional transformer models such as ELMo can only focus on the features in one direction. There is a lack of large-scale specialized corpora in the field of cybersecurity. Using the pretrained BERT model, the prior knowledge learned in the pretraining stage can make up for this defect to a certain extent. At the same time, the multilayer transformer coding of BERT helps to address the issue of polysemy in cybersecurity entities. When different sentences containing the same word are input into BERT, the word vectors outputted by BERT will be different, allowing for a more accurate representation of the intended meaning of the word.

BERT embedding first converts the input text into token sequence through WordPiece tokenization and adds a CLS tag at the beginning and a SEP tag at the end of the sentence. In addition to the strong representation ability brought by BERT structural design, the applicability of the BERT WordPiece tokenization method to security texts is also taken into consideration. The effect of the WordPiece model [42] based on language modeling is to split words into fragments. By separating the meaning of words from the changes of affixes and tenses, the WordPiece model helps effectively reduce the number of words in the dictionary. For example, the word "upload" in the input text will be split into "up" and "##load," and the word "WindowsXP" will be split into "Windows," "##X," and "##P." As mentioned above, security texts have the characteristics of nonstandardized naming and frequent neologisms. The traditional method segments sentence and forms the vocabulary by counting and selecting the top N words with the highest frequency, which cannot perform well on security texts for it usually cannot contain all the words in the security data set and lead to many out-of-vocabulary (OOV) words. Also, the low-frequency words cause word features high-dimensional and sparse, and words in different tenses will be treated as different words which increases training redundancy. The subword tokenization model breaks down words into meaningful subwords, which can solve the problems mentioned above and find a balance between the vocabulary size and the unknown word coverage.

4.3.2. Attention-Based BiLSTM-CRF NER Model. Section 4.2.1 has described the advantages and applicability to our dataset of attention-based BiLSTM model. The difference is that the NER model adds an additional conditional random field (CRF) layer to increase the consideration of label information. Although the attention-based BiLSTM can be directly used for sequence annotation, it ignores the consideration of the label information, take "CVE-2019-9810 has been exploited at Pwn2Own 2019" as an example. The label of "CVE-2019-9810" output by LSTM is "noun," and the label of "Pwn2Own" may also be "noun." But in fact, "CVE-2019-9810" label is "vulnerability" and the "Pwn2Own 2019" label is "security term," that is, the rule of "vulnerability" + "security term" is not captured by the model. Therefore, the conditional random field model is introduced to the NER model.

For the conditional random field, its state depends on its adjacent state. In the linear model, it depends on the previous state and the latter state. It can be expressed as (6), where $X(t_{n-1}) = x_{n-1}, X(t_{n-2}) = x_{n-2}, \ldots, X(t_1) = x_1$ represents the state of the first $n-1$ step in the sequence. The $X(t_n) = x_n$ represents the status of step n. The probability of status $X(t_n) = x_n$ is interrelated to step $n-1$ and step $n+1$.

$$\begin{aligned}\left(X(t_n) = x_n \middle| X(t_{n-1}) = x_{n-1}, X(t_{n-2}) = x_{n-2}, \ldots, X(t_1) = x_1\right) \\ = P\left(X(t_n) = x_n \middle| X(t_{n-1}) = x_{n-1}, X(t_{n+1}) = x_{n+1}\right).\end{aligned} \tag{6}$$

The performance effect of conditional random field alone is not satisfactory. Combining it with a deep learning model can enhance overall performance. First, the input Readme text is transformed into word embedding using a pretrained BERT model. These embedding are then used as input to the attention-based BiLSTM neural network for feature extraction. The BiLSTM layer captures contextual features and generates hidden state representations, denoted as h_t. The attention mechanism assigns different weights to the contextual words. Then, the CRF layer calculates the probability distribution of labels for each word based on the output of the attention layer. According to the label transfer matrix of CRF, the score of label Y_{i-1} of the previous word and label Y_i of the current word can be obtained. Finally, the label with the highest probability score is assigned as the sequential labeling prediction for the current word.

4.4. Relationship Extraction. After the entity classification and entity recognition module completes the recognition of entities, it extracts the relationships between entities. The relationship extraction method is rule-based and can be divided into two aspects: regular expression-based rules and ontology-based rules. The regular expression-based relationship extraction is performed during the data acquisition process. The relationships of use, belong_to, and located_at relationships and the associate relationship between SecUser entities are known relationships in the open-source community and can be matched based on regular expressions. Taking the use relationship as an example, we match the users who perform star, fork, and follow actions

on security tools by crawlers. These users have a use relationship with the tools. For other relationships, a set of rules based on the security tools ontology are defined to establish relationships between entities.

The ontology-based relationship extraction rules rely on the premise that security tools only refer to relevant entities. According to our observation of the characteristics of security tools' data, the tools provide concise descriptions of the usage and features in the Readme, while rarely mentioning unrelated information or unsupported function. Therefore, if two entities of the same security tool are extracted and the types of these two entities have a defined relationship in the ontology, then a unique relationship between the two entities exists. Specifically, when an open-source tool is classified as a security tool and is defined as sectool_i, the attack phase entity phase_i for sectool_i can be obtained through the page-level entity classification model, and the consequence entity consequence_i and type entity type_i can be obtained through the paragraph-level entity classification model. On the basis of the security tools ontology, the relationship among $(\text{sectool}_i, \text{implement}, \text{phase}_i)$, $(\text{sectool}_i, \text{cause}, \text{consequence}_i)$, and $(\text{sectool}_i, \text{associate}, \text{type}_i)$ can be obtained. Assuming that the developer of this security tool is secuser_i, the secuser_i has implemented relationship with phase_i. Through the NER model, the vulnerability entity vul_i and four types of attributes of sectool_i can be recognized. According to the ontology, the sectool_i has exploit relationship with vul_i, and vul_i has cause relationship with consequence_i. The security term, protocol, operating system, and hardware attributes have associate relationship with sectool_i.

5. Experiment

5.1. Dataset. Since the dataset used in the experiment is the newly labeled data set in this study, we will introduce the data source, data labeling, and label distribution of the experimental data set in this section.

5.1.1. Data Source. This experiment uses GitHub as the data source, considering that GitHub is the largest open-source community platform at present with good community ecology and comprehensive and diverse data types.

We employed two methods for data acquisition: a collection repositories-based method for one-time updates and a seed user-based method for incremental updates. Some repositories, such as [9], collect security-related tools on GitHub. The collection repositories-based data acquisition method is to collect tools and users' information from those collection repositories. We crawled 12 repositories that collected security tools, extracted the GitHub repository links, and then proceeded to crawl tools and related users data from these repositories. In addition, we manually selected a set of high-quality seed security users, totaling 903 users, who demonstrated active engagement and a strong interest in security-related projects on GitHub. To ensure ongoing data updates, we periodically crawled their follow or star lists to capture any changes. By employing these combined strategies, we obtained a comprehensive and up-to-date dataset for our research on security tools and projects. With the help of the official API provided by GitHub and the requests library of Python, complete information on these tools and users is obtained. At last, more than 40 thousand tool repositories' information and more than 3 million users' information are crawled, including their relevant attributes. The entity type, corresponding quantity, and various entity attributes that crawled are shown in Table 6.

5.1.2. Data Labeling. We manually marked some of the data for the training of the deep learning model. Two lab-mates are invited to perform the annotation. Both of them have a bachelor's or higher degree in cyber security and equip in-depth knowledge of software engineering and security. For labeling the data used in the classifier, we manually assigned a label to each security tool based on its topics, descriptions, and Readme text according to the category definition. The specific dataset volume involved in model training and evaluation is shown in Table 5, and the detail of classifier label distribution is shown in Table 4. For NER labeling, label each word in the security tools Readme text by assigning one of the six labels: vulnerability (VUL), security term (ST), protocol (P), operating system (OS), hardware (HW), or others (O). Those words that are not related will be labeled as O. The BIO annotation strategy is adopted for the labeling. If the entity is a word combination, the first word is marked as "B-" entity type (e.g., B-OS), and the subsequent words are marked as "I-" entity type (e.g., I-OS).

5.2. Experimental Design. In this section, the superiority of our knowledge extraction method will be proved by separately evaluating the effectiveness of classifiers and the named entity recognition model. The open-source tools' dataset is divided into training, test, and validation sets with an 8:2:2 ratio. The validation set is used to evaluate the models. Based on the evaluation matrix, the effect is evaluated from three aspects of precision, recall, and $F1$-score. The hyperparameter settings of our knowledge extraction model are shown in Table 7.

5.2.1. Evaluation Matrix. The evaluation criteria adopted in this paper are consistent with the prevailing standards at home and abroad. Researchers usually use the precision, recall, and $F1$ score as the evaluation means. This study also adopts these three standards to evaluate the performance of the model. The calculation methods are shown in equations (7)–(9). Generally, TP is used to represent the number of samples that predict the positive class as the positive class, FN is used to represent the number of samples that predict the positive class as the negative class, and FP is used to represent the number of samples that predict the negative class as the positive class.

Table 6: Crawled raw dataset.

Name	Number	Attributes
Repositories	47,440	repo_id, description, topics, Readme, branches, homepage, license, owner_type, programming language, and created_at, updated_at
Users	3,910,704	Username, user_id, avatar, user_type, actual_name, company, blog, e-mail, hireable, twitter, created_at, and updated_at
Organizations	8,058	organization_name, organization_id, avatar, description, blog, e-mail, twitter, created_at, and updated_at
Issues	1554,586	issue_id, title, body, and open_state

Table 7: Hyperparameter settings of our knowledge extraction model.

Hyperparameter	Value
Epochs	30
Batch_size	64
Dropout rate	0.4
Optimizer	Adam
Kernel size	3
Units	128
Kernel_regularizer	L2

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{7}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{8}$$

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{9}$$

5.2.2. Page-Level Classification Model for Security Tool's Attack-Phase Extraction. This model is the most important model in our system, which is used to distinguish security-related tools from massive open-source tools and classify security tools from the perspective of attack. Page-level-basedattack-phase multiclassifier model takes the full Readme text of the tool as the input text and outputs the result of one of the five attack stages or the result of non-related tools. By classifying the attack-phase of tools, potential attack-related tools can be provided for security professionals from the perspective of attackers, avoiding the disadvantage of a single perspective directly classified as security defense tools.

When evaluating this classifier, we design to prove the effectiveness of this method through the following comparative experiments: feature selection and neural network selection. The feature selection aims to prove that just choosing a tool's topics, description, or first paragraph may not represent the security tools attack-phase properly. Hence, this experiment compares the effect of different input text as follows: topics and description, topics and first paragraph, and full Readme text. In Figure 3, the horizontal axis represents different text features used as inputs, while the vertical axis represents the precision of the attention-based BiLSTM model and TextCNN model when using that feature as input. When the embedding method and

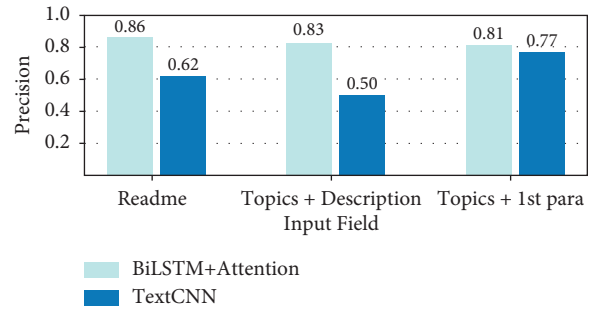

Figure 3: Impact of choosing different text fields and neural networks on page-level classification for security tool's attack-phase extraction.

classification model remain unchanged, although on the training set the performance is similar, page-level field selection has obvious advantages in the validation set. As shown in Figure 3, the precision of taking full Readme text as input is improved by 3–5% than other selections when the attention-based BiLSTM model uses default parameters. We compared three commonly used models, namely RNN, LSTM, and TextCNN, with BiLSTM and attention mechanism to demonstrate the effectiveness of BiLSTM and attention mechanism on our long input data. As shown in Table 8, the precision of attention-based BiLSTM model is improved by 9% than the RNN model and LSTM model. This also formalizes our analysis in the previous section; attention-based BiLSTM model can build long-distance dependence between texts and capture the import information features at the same time. For the TextCNN model on page-level classification, efforts have been made to raise the final performance of the model, specifically, adjusting the structure of the convolutional layer, pooling layer, the size of kernel size, loss function, and activation function. Despite these attempts to adjust, the TextCNN model continues to underperform in accurately identifying the attack-phase of open-source tools. In comparison to the attention-based BiLSTM model, the precision of the TextCNN model is more than 20% lower.

Furthermore, we compared our proposed paragraph-level model (BiLSTM and attention mechanism) with relevant tools used in previous studies, including SourceFinder [8], TIMiner [43], and SecureBERT [44]. All of them are based on supervised learning to classify unstructured text into security-related categories. SourceFinder classifies Github repositories as malware or other based on fields such as name, Readme, and description. TIMiner collects security

Table 8: Results for page-level and paragraph-level classification.

Model	Class	Word embedding	Neural network	Precision (%)	Recall (%)	$F1$-score (%)
Page-level	Attack-phase	Word2Vec	RNN	77.29	67.08	70.86
			LSTM	77.41	70.42	73.21
			TextCNN	61.97	56.34	59.05
			BiLSTM + attention	86.33	85.56	85.89
Paragraph-level	Type	Word2Vec	RNN	83.56	79.91	81.70
			LSTM	83.92	82.09	82.99
			TextCNN	85.26	82.65	83.93
			BiLSTM + attention	82.14	81.78	81.95
	Consequence	Word2Vec	RNN	79.32	75.58	77.36
			LSTM	85.54	83.04	84.22
			TextCNN	89.29	85.27	86.99
			BiLSTM + attention	86.90	82.73	84.68

text from security blogs and reports to classify them into five impact domains. SecureBERT is a language model designed for cybersecurity that can capture the complex semantics of security text. Orbinato et al. [45] proposed an automated method to map CTI reports to ATT&CK techniques, and the SecureBERT-based classification model achieved the best performance among 11 models in the comparison experiment. We fine-tuned three models on our security tools dataset, and the experimental results are shown in Table 9. It appears that the traditional machine learning algorithms used in SourceFinder cannot capture the security categories expressed in the text semantics, resulting in a low evaluation precision on the validation set. The SecTKG page-level classification model achieved an $F1$ score that is 23% and 13% higher than TIMiner and SecureBERT, respectively. It should be noted that SecureBERT suffered from severe overfitting in the experiments, which may be due to the limited size of our labeled dataset. We attempted various methods to alleviate overfitting, including only adjusting the last hidden layer of the model during training, increasing the dropout rate, and applying L2 regularization, but the highest precision we could achieve was only 74%.

Through the comparative experiments on feature selection and model selection, we have demonstrated the rationale for classifying the attack-phase of tools at the page level, as well as the ability of the attention-based BiLSTM model to extract features from long texts. Furthermore, compared to more complex deep learning models such as BERT, the attention-based BiLSTM model is better suited for mitigating overfitting issues in limited dataset scenarios.

5.2.3. Paragraph-Level Classification Model for Security Tool's Type and Consequence Extraction. If an open-source tool is classified as one of the attack-phase after the page-level classification model illustrated in Section 4.2, take the tool as a security tool and mine more characters of it. This model is used to recognize whether the security tool is system type or application type and what consequence the security tool may cause.

As we already get the attack-phase of the security tool after the page-level classification, take the advantage of this feature and add the crawled programming language and topics' features to the input text to improve the effectiveness of the classifier. Three groups of different inputs are set up to compare the classification effect. They are (a) topics + description, (b) full Readme text, and (c) attack-phase + programming language + topics + first paragraph of the Readme. The x-axis of Figure 4 represents three feature groups mentioned above, while the y-axis represents the precision of classifying type and consequence categories when selecting a particular feature group as input. As shown, when using the same neural network, the (c) feature combination has the best performance, with a maximum improvement of 16% in precision.

Adjusting the TextCNN model parameters to better capture the text features, results come out that the convolutional layer below three layers is difficult to extract effective features from the text, and the feature extraction effect is the best when the convolution kernel size is 3. The overfitting problem can be alleviated by increasing L2 regularization loss. The effect of our paragraph-level model is also shown in Table 8. The experiment compares the performance of the RNN model, LSTM model, and attention-based BiLSTM model on the tool's type and consequence classification. After parameter adjustment and optimization, the TextCNN model can do better with 3% than the attention-based BiLSTM model, and the training time of each round of the model can be reduced by 90%. Besides, under the condition of approximate training time, the TextCNN model can do better with 2–9% than the RNN model. The more complex neural network and longer training time of the attention-based BiLSTM model did not get a better classification effect on the security tool's type and consequence extraction. We considered that the length of the input text is short, and there is no obvious word order relationship in the spliced feature fields which leads to this result. Table 9 shows the comparison results of SecTKG paragraph-level classification model and SourceFinder, TIMiner, and SecureBERT. Our improved TextCNN model outperforms TIMiner with an increased accuracy of 7.95% for the type classification and 5.82% for the consequence classification. SourceFinder's accuracy has slightly improved in the type binary classification but still performs poorly in the consequence multiclass classification. The SecureBERT model also suffers from overfitting issues in paragraph-level classification as mentioned in the previous section, which

Table 9: Classification method comparison between SecTKG and SourceFinder, SecureBERT, and TIMiner.

Class	Model	Neural network	Precision (%)	Recall (%)	$F1$-score (%)
Attack-phase	SourceFinder (2020) [8]	Multinomial Naive Bayes	18.03	5.72	8.7
	TIMiner (2020) [43]	CNN	70.86	55.97	62.40
	SecureBERT (2022) [44]	SecureBERT	74.33	71.34	72.35
	SecTKG_Page-level	BiLSTM + attention	86.33	85.56	85.89
Type	SourceFinder	Multinomial Naive Bayes	69.53	73.51	71.46
	TIMiner	CNN	77.31	60.53	67.89
	SecureBERT	SecureBERT	84.25	91.91	87.92
	SecTKG_Paragraph-level	TextCNN	85.26	82.65	83.93
Consequence	SourceFinder	Multinomial Naive Bayes	16.3	23.57	19.27
	TIMiner	CNN	83.47	63.68	72.20
	SecureBERT	SecureBERT	86.73	85.92	85.79
	SecTKG_Paragraph-level	TextCNN	89.29	85.27	86.99

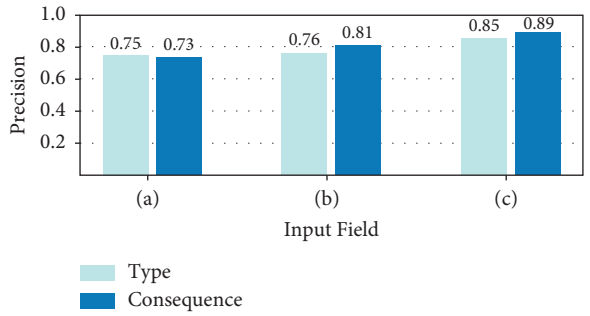

Figure 4: Impact of text field chosen for paragraph-level classification: (a) topics + description, (b) full Readme text, and (c) attack-phase + programming language + topics + first paragraph of the Readme.

results in a decreased accuracy on the validation set. By means of experimental comparisons of feature selection and model selection, we have demonstrated that SectKG paragraph-level classification based on TextCNN model outperforms the baseline model and state-of-the-art research in the field.

5.2.4. Named Entity Recognition Model for Irregular Attributes' Extraction. This experiment compared the impact of two word embedding models, BERT and Word2Vec, on the performance of NER model. The BERT model is using the model trained by Jackaduma [46] on the cybersecurity corpus and the Word2Vec model is using the model trained by Youngja [47] on the cybersecurity corpus. The hidden units, batch size, optimizer, and learning rate are set to 128, 32, Adam, and 0.001.

The effect of the NER model is shown in Table 10. Compared with Word2Vec word embedding, the BERT embedding F1-score does better with about 5%. As already mentioned in Section 4.3, BERT embedding has advantages in its network structure and tokenization model, which we think is the reason why BERT embedding has better performance than Word2Vec embedding on the security tools' dataset. The recall and $F1$-score of the BiLSTM effect has improved by 2.6% and 1.1% compared with BiGRU, indicating that the combination of attention mechanism and LSTM + CRF model can better identify network security named entities.

Table 11 shows the experimental results of entity recognition models used by four tools, namely Open-cykg [12], CSKG4APT [14], CTI View [48], and Vulcan [49], on the security tool dataset. These tools are all used to identify security entities from security texts. Open-cykg incorporates a time-distributed dense layer into the entity recognition model to build a BiGRU-ATT-TDD-CRF model for identifying open CTI knowledge. CSKG4APT and CTI View both use a BiLSTM-GRU-CRF model to identify APT-related knowledge from CTI corpus. Vulcan adopts a BiLSTM-CRF model to identify CTI knowledge from multiple sources of data. As shown in Table 11, the word embedding method and neural network utilized by SecTKG were superior to other models. We attribute this to two main reasons. Firstly, SecTKG employed a BERT model trained on security corpora, which was more effective in embedding security terms than BERT models trained on general domains. Secondly, the attention layer was added to allow the model to focus more on key features. Although the open-cykg tool also added an attention layer, considering that our entities did not have any significant time span differences, the added TDD layer in open-cykg may have affected the final output.

6. Results and Discussion

6.1. Data Analysis. After the knowledge extraction experiment, we analyzed the experiment results and explored the distribution and development trend of security tools. In addition, we found some security tools' characteristics different from general open-source tools by comparing the statistical results with some public reports [50, 51].

6.1.1. Distribution of Tools. At last, 15,778 security tools are identified from the 35,214 repositories we collected. Some statistics are made according to the security tools ontology. First, the attack-phase of security tools is analyzed. As shown in Figure 5, security tools belong to the exploit phase accounting for the largest proportion, and security tools belong to the actions phase accounting for the minimum proportion. Specifically, 5,010 security tools belong to the

Table 10: Comparison of different word embedding and neural networks of the named entity recognition model.

Embedding	Neural network	Precision (%)	Recall (%)	$F1$-score (%)
Word2Vec	LSTM + CRF	92.47	70.22	79.47
	CNN + LSTM + CRF	97.41	78.51	86.86
	BiGRU + attention + CRF	96.92	78.88	86.85
	BiLSTM + attention + CRF	97.04	81.81	87.59
BERT	LSTM + CRF	97.58	71.11	82.10
	CNN + LSTM + CRF	98.14	85.01	91.09
	BiGRU + attention + CRF	98.33	84.95	91.14
	BiLSTM + attention + CRF	97.85	87.33	92.28

reconnaissance phase, accounting for 31.75%. 5,329 security tools belong to the exploit phase, accounting for 33.77%. 1,824 security tools belong to the persistence phase, accounting for 11.56%. 3,154 security tools belong to the lateral movement phase, accounting for 19.99%. 461 security tools belong to the actions phase, accounting for 2.92%. Second, according to the identification results of the paragraph-level classification model, the types and consequences of security tools are counted. For security tools type, only 1,492 security tools work on the system level, and the rest all belong to application tools. For the consequence that will be caused by these security tools, 1,096 tools will cause DenialOfService consequence, 3,438 tools will cause LossOfConfiguration consequence, 7,416 tools will cause LossOfInformation consequence, and 3,828 tools will cause RemoteAccess consequence.

6.1.2. Popular Programming Languages. According to the knowledge extraction framework, the corresponding programming language attributes are extracted from 15,778 identified security tools. Comparing the results with the popular programming languages of general open-source tools to explore whether security developers have different preferences for programming languages, the state of the octoverse 2022 [50] shows the most used languages, and the top 10 languages are JavaScript, Python, Java, TypeScript, C#, C++, PHP, Shell, C, and Ruby. According to our statistics on the programming language of security tools, the top 10 popular programming languages are Python (30.78%), C (11.46%), Go (9.73%), C++ (7.23%), JavaScript (5.53%), Shell (5.2%), C# (3.75%), Rust (3.5%), Java (3.17%), and Powershell (2.54%). It can be found that the most commonly used language for security developers is Python rather than JavaScript, which has an overwhelming advantage in the overall proportion. At the same time, the use of C, shell, and PowerShell by security developers is also significantly higher than that of other developers.

Furthermore, an analysis was done to figure out whether the abovementioned phenomenon is owing to a different attack phase has a different preference for programming languages. The outcome shows that Python and Go languages are frequently used in the reconnaissance phase, which may be due to the ease of use of the language and the sound third-party library. C, C++, Shell, and PowerShell are used more frequently in the exploit phase and persistence phase, which have an advantage in operating efficiency and higher system relevance. For the lateral movement phase, Python, Go, C/C++, and JavaScript are more popular. For the action phase, Python and C are more popular.

6.1.3. Popular Operating Systems. The aim of this analysis is to explore whether security tools are biased towards different operating systems, or which operating system security developers pay more attention to. Based on our named entity recognition model, 12,492 operating systems related to security tools are recognized. The statistical result shows that Linux is the most popular operating system for security developers, which account for 58.58%. Windows, macOS, Android, and IOS account for the proportion 21.08%, 9.18%, 4.8%, and 2.19%, respectively. However, it seems that there are no statistics on the usage proportion of different operating systems in the open-source community to date. For reference, comparisons are made with StatCounter's "operating system market share worldwide—March 2022" [51] and Thomas Alsop's statistic for the global server operating system market [52]. According to StatCounter statistic, the top 5 operating systems and respective proportions are Android (41.59%), Windows (31.12%), IOS (16.87%), OS X (6.3%), and Linux (0.97%). And Thomas found that Windows and Linux are the most popular operating system for servers, whose proportion ratios are 72.1% and 13.6%, respectively. It is obvious that the Linux operating system is widely used by security developers. Besides, Ubuntu, Debian, and Kali are the top 3 popular Linux operating systems in the security tools' dataset. Compared to Ubuntu and Debian operating systems, the correlation between Kali operating system and security is more obvious. Kali contains more than 600 penetration testing tools, which are suitable for various information security tasks.

6.1.4. Vulnerability Entities' Temporal Analysis. Based on our named entity recognition model, 10,542 vulnerability entities are recognized. The purpose of this analysis is to get a feel for these vulnerability entities' distribution and evaluation trends. Considering that CVE (common vulnerabilities and exposures) numbers have a clear date and are used widely, select the vulnerability entities with CVE numbers for the convenience of statistics. 2,815 different CVE numbers were found in the security tools. The process of data collection was continued as of March 2023, 68.12% of CVE numbers were in the past six years, and the remaining were irregularly distributed from 1999 to 2016. This result shows that the vulnerabilities involved in the security tool

Table 11: Named entity recognition model comparison between SecTKG and the state-of-the-art models.

Tool	Word embedding	Neural network	Precision (%)	Recall (%)	*F*1-score (%)
Open-cykg (2021) [12]	XLM-RoBERTa	BiGRU + attention + TDD + CRF	95.72	78.66	86.26
CSKG4APT (2022) [14] and CTI view (2022) [48]	BERT	BiLSTM + GRU + CRF	95.90	77.79	85.69
Vulcan (2022) [49]	BERT	BiLSTM + CRF	97.45	76.02	85.16
SecTKG	Cybersecurity BERT	BiLSTM + attention + CRF	97.85	87.33	92.28

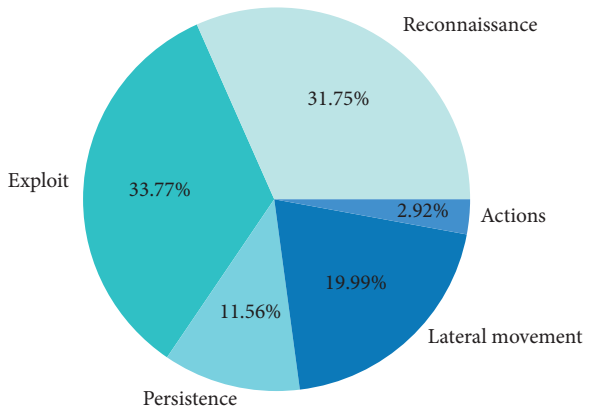

Figure 5: Attack-phase distribution of security tools.

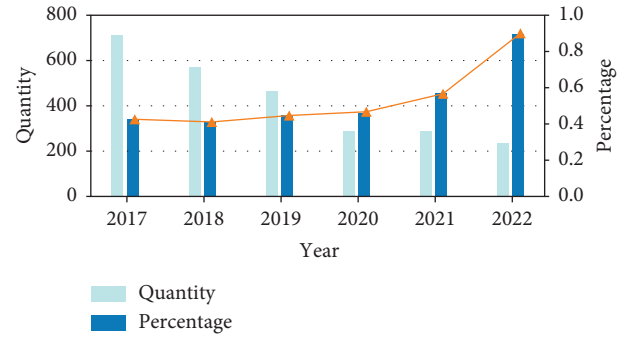

Figure 6: The proportion of security tools involving the latest vulnerabilities increases year by year.

are relatively new. The CVE reported years from 2017 to 2022 are carefully counted, and the relationship between their creation time and vulnerability year is compared. As shown in Figure 6, x-axis represents different years; the light blue color represents the number of recognized vulnerability entities with CVE numbers in this year, while dark blue color representing the proportion of security tools involves the latest CVE numbers in this year. More than 40 percent of the vulnerability numbers created relevant security tools in the same year. The proportion increased year by year, from about 42.53% in 2017 to 89.70% in 2022. The data collection of security tools will keep going in the future to count the vulnerabilities' distribution changes in 2023 and after.

It is meaningful and necessary to identify and analyze vulnerable entities in security tools. The timely detection and identification of security tools in the open-source community can help security professionals master the latest vulnerability-related security tools. Further services for attack detection, traceability, and defense are provided.

6.1.5. Security Developers. According to the identified security tools, the information of up to 2.6 million relevant security developers or organizations in the open-source community was collected. The insights into security developers' characteristics will be discussed from three aspects: the location of developers, employment status, and the number and update frequency of public repositories.

(i) Location of security developers: A total of 922 thousand location information related to security developers and organizations are collected. These developers' location distribution statistics show that the top 5 popular countries are China, Brazil, India, Germany, and Canada. Moreover, China's top 8 popular cities are Beijing, Hangzhou, Shanghai, Shenzhen, Chengdu, Guangzhou, Nanjing, and Wuhan. These eight cities bring together 52.12% of security developers in China. This phenomenon is likely because these cities have a large number of colleges and companies.

(ii) Employment status: A total of 209 thousand employment statuses related to security developers are collected. Only five percent of them were in university, which is not as high as one may surmise. The small proportion of students may result from incomplete registration information. Nevertheless, much information about developers has been collected. For instance, work companies, personal blogs, and Twitter accounts, which would be further helpful in understanding the personal situation of security developers.

(iii) Public repositories' numbers and update frequency: For repositories identified as security tools, count the number of public repositories owned by these developers. Figure 7 shows the distribution of security developers at different intervals. Each bar represents a range of 5, taking the first bar as an example, 1,538 security developers created repositories within the range of 0–5. The result shows that more than half of the security developers have more than 25 public repositories. In addition, the update frequency of security developers is observed. According to the time when the data collection stops, 73.1% of security tools' developers have updated the project in nearly three months. To some extent, these two attributes of security developers reflect the community contribution of security developers and the possibility of security tool maintenance.

6.2. Knowledge Graph Establishment. This module is realized with the Neo4j browser. The Neo4j browser allows developers to execute Cypher queries and visualize the results. Our system used the configuration of Neo4j-community-3.5.1 and JDK 1.8.

Overall, 4 million entities and 10 million relationships are extracted. By importing these entities, entities' attributes, and relationships' information into the Neo4j database in the form of $(\text{entity}_i, \text{relationship}, \text{entity}_j)$ or $(\text{entity}_i, \text{attribute}_j, \text{attributevalue}_j)$ triples, queries can be made on any node in the security tools' knowledge graph with Cypher statement on the web page. Neo4j browser uses circles to represent nodes and lines to represent relationships. The defined node attributes can be displayed in tabular form through the query. Figure 8 shows an example of query results. The

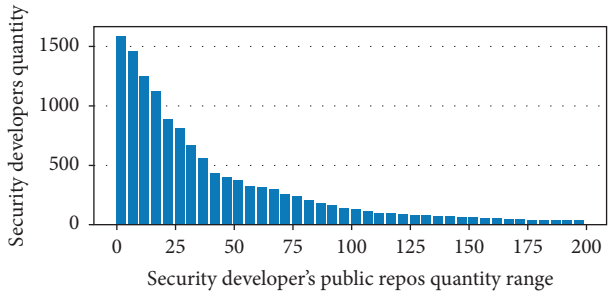

Figure 7: Distribution of security developers' public repositories.

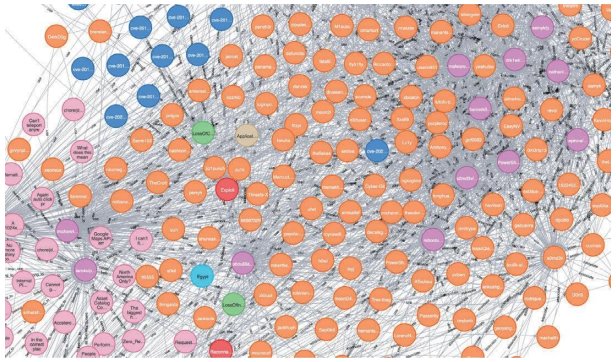

Figure 8: A visualization sample of SecTKG. The purple, orange, red, green, brown, dark blue, light blue, and pink nodes represent SecTool, SecUser, attack-phase, consequence, type, vulnerability, location, and issue entity types.

purple, orange, red, green, brown, dark blue, light blue, and pink nodes in Figure 8 represent SecTool, SecUser, Attack-phase, consequence, type, vulnerability, location, and issue entity types. The connection between two nodes represents the relationship between two entities, and the text on the connection indicates the type of relationship. In the future, the visual website of the SecTKG will be published to provide online search services for security professionals.

6.3. Identifying Influential Repositories. Based on the constructed security tools' knowledge graph, we realize a practical application for measuring security tools' influence. Through the analysis of the identified security tools, it is proved that the application can indeed find some high-quality but uncommon security tools.

6.3.1. Security Tools' Influence-Measuring Method. After building the knowledge graph base on the security tools' ontology proposed in Section 3, the knowledge graph can be used in practice by applying some graph algorithms. Accordingly, we implement a method for security tools' influence measuring based on the multiattribute decision-making and linear threshold (LT) propagation algorithm. This method considers node attribute information, different types of relationships, and social network analysis and aims to find some security tools which are high-quality, well maintained, but less famous. Avoid the lack of ranking of open-source tools based on a single relationship, such as star or fork. The experimental analysis shows that this method does find some rarely used but high-quality security tools. The detail of security tools' influence-measuring method is as follows:

Our propagation algorithm will traverse the tool nodes in the network instance in turn. For any tool node $t_i \in T$ (tool set), activate all user nodes connected to t_i, and the other nodes are inactive. Using b_{wv} indicates that node v is affected by its neighbor node w. b_{wv} is defined as the calculation (10), where s_{t_i} represents the quality score of security tool t_i and w_{wv} represents weight of edge between node w and node v. $in(w)$ means the direction of the edge is the node pointing to node w. Node u belongs to the node set which links to node w.

$$b_{\mathrm{wv}} = \begin{cases} \sum_{u \in \mathrm{in}(w)} b_{\mathrm{uw}} \cdot \left(1 + s_{t_i}\right) \cdot w_{\mathrm{wv}}, & u \neq t_i, \\ \left(1 + s_{t_i}\right) \cdot w_{\mathrm{wv}}, & u = t_i. \end{cases} \tag{10}$$

The quality score of the security tool s_{t_i} is calculated by the multiattribute decision-making method TOPSIS [53]. TOPSIS uses the distance between the evaluation object and the optimal solution and the worst solution to judge the quality of the evaluation object. The quality score of the tool can be obtained without preset attribute preference. We assumed that except the number of star or watch of open-source tools, the description documents and post-maintenance reflect the quality of the tools to some extent. For example, a qualified GitHub repository usually has a complete Readme file and not one-off. Hence, six features are selected to represent the quality of the tools. They are (a) the length of Readme, (b) if the tool has description information, (c) if the tool has open issues, (d) if the Readme follows markdown paragraph specifications, (e) if the owner introduces the tool function in Readme, and (f) if the owner maintains the repository after the first commit. Based on these features, we turned tool attributes into feature evaluation metrics. Then, calculate the distance of each vector to the optimal solution and the worst solution; the closeness between the evaluation object and the optimal solution is the quality score of tool s_{t_i}.

When the sum of the influence of the neighbor nodes of node v is greater than the threshold of node v, node v is activated, that is, the cumulative influence of the activated neighbor nodes of node v on v is greater than the threshold θ_v of node v.

$$\sum_{w \in \mathrm{in}(v), \mathrm{active}(w) \neq 0} b_{\mathrm{wv}} \geq \theta_v. \tag{11}$$

After node v is activated, it will affect its neighbor nodes the next time and repeat the abovementioned process. When the sum of the influence of all existing active nodes in the network cannot activate the inactive neighbor nodes, the propagation process ends. Define the number of activated nodes as the final node influence score.

Table 12: Popular influence SecTools.

Category	SecTool
Reconnaissance	bettercap, subfinder, torsniff, AutoRecon, and gping
Exploit	ncrack, SecLists, wifi-cracking, routersploit, and Java-Deserialization-Cheat-Sheet
Persistence	webshell, shellcheck, fabric, PowerSploit, and xonsh
Lateral movement	trpc, bash-it, winston, localtunnel, and Keylogger
Actions	MHDDoS, poisontap, dnscat2, evil-winrm, and botnet-hackpack

Table 13: Details of popular influence SecTools.

Category	SecTool repository	URL
Reconnaissance	bettercap/bettercap	https://github.com/bettercap/bettercap
	projectdiscovery/subfinder	https://github.com/projectdiscovery/subfinder
	fanpei91/torsniff	https://github.com/fanpei91/torsniff
	Tib3rius/AutoRecon	https://github.com/Tib3rius/AutoRecon
	orf/gping	https://github.com/orf/gping
Exploit	nmap/ncrack	https://github.com/nmap/ncrack
	danielmiessler/SecLists	https://github.com/danielmiessler/SecLists
	brannondorsey/wifi-cracking	https://github.com/brannondorsey/wifi-cracking
	threat9/routersploit	https://github.com/threat9/routersploit
	rrrDog/Java-Deserialization-Cheat-Sheet	https://github.com/rrrDog/Java-Deserialization-Cheat-Sheet
Persistence	tennc/webshell	https://github.com/tennc/webshell
	koalaman/shellcheck	https://github.com/koalaman/shellcheck
	fabric/fabric	https://github.com/fabric/fabric
	Mafia/PowerSploit	https://github.com/Mafia/PowerSploit
	xonsh/xonsh	https://github.com/xonsh/xonsh
Lateral movement	trpc/trpc	https://github.com/trpc/trpc
	Bash-it/bash-it	https://github.com/Bash-it/bash-it
	winstonjs/winston	https://github.com/winstonjs/winston
	localtunnel/localtunnel	https://github.com/localtunnel/localtunnel
	GiacomoLaw/Keylogger	https://github.com/GiacomoLaw/Keylogger
Actions	MatrixTM/MHDDoS	https://github.com/MatrixTM/MHDDoS
	samyk/poisontap	https://github.com/samyk/poisontap
	iagox86/dnscat2	https://github.com/iagox86/dnscat2
	Hackplayers/evil-winrm	https://github.com/Hackplayers/evil-winrm
	TreeHacks/botnet-hackpack	https://github.com/TreeHacks/botnet-hackpack

$$\text{Influential SecTool Rank} = \sum_{\text{active node}} \text{Rank}. \tag{12}$$

6.3.2. Influential Repositories' Analysis. According to the page-level classification model for the security tool's attack-phase, the collected tools are classified into different categories. Calculate the influence ranking of tools for each attack-phase and analyze the results. Table 12 shows the top 5 influence tools for each attack-phase based on our security tools' influence-measuring method. We compare these 25 security tools with the Kali tools' list and found 21 tools not included in the latest Kali tools' list. We find that although Kali has comprehensive coverage of tools in the reconnaissance phase, it has poor coverage in the lateral movement phase and action phase security tools. Observing that some of these security tools are unfamiliar to us, we list repositories' full name and access URL in Table 13, which can be accessed to learn about these tools.

7. Limitations

It should be noted that the five categories of security tools identified by us do not cover defense tools such as firewalls or intrusion detection systems. Currently, SecTKG only identifies tools that belong to the five attack stages of reconnaissance, exploit, persistence, lateral movement, and actions. Defense tools, tutorials, and tools unrelated to security are classified as others. In addition, due to a lack of relevant research, there is no public dataset available for evaluating the effectiveness of our knowledge extraction approach for security tools. Thus, we can only compare our results with similar studies that used datasets constructed for tasks with similar characteristics. However, due to differences in the datasets and recognition tasks, these comparative models often fail to achieve the same level of accuracy reported in their original papers when applied to the security tool dataset.

8. Conclusion and Future Work

In this work, we conduct the first study on how to build a large-scale knowledge graph for open-source security tools. A security tools' ontology model is designed to portray security tools' application scenarios, technical characteristics, popularity, related security users, and their associations. Different knowledge extraction methods are designed according to the characteristics of open-source security tools and demonstrated the effectiveness of SecTKG knowledge extraction methods by comparing them with multiple baseline and state-of-the-art models. The analysis of experimental results yielded valuable insights into open-source security tools. In addition, we realized a practical application for measuring the influence of security tools, which proved capability of identifying some high-quality but uncommon security tools. SecTKG associates security tools' features, vulnerabilities, consequences, and relevant users into a whole graph, facilitating the tracing of relevant security tools and attackers from fragmented attack information. Our work fills a void in research on open-source security tools and provides a foundation for security professionals to conduct forensics and traceback.

In future work, we plan to enhance the granularity of the knowledge extraction by considering the issues of nested entity and discontinuous entity recognition, in addition to the current entity recognition methods in the paper. We also aim to expand the scope of the security tools' collection and develop a knowledge base of open-source security tools from multiple open-source community platforms. In addition, we will investigate knowledge merging and inference techniques for the SecTKG, exploring ways to mine similarities between security tools, identify community divisions, and analyze interactions among security users.

Data Availability

The dataset is not publicly available directly due to the security issue of attack tools. Data are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research is funded by the National Key Research and Development Program of China (No. 2021YFB3100500), National Natural Science Foundation of China (No. 61902265), CCF-NSFOCUS KunPeng Research Fund (No. 202105), Open Fund of Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation (No. CSSAE-2021-001).

References

[1] Splunk, "The state of security," 2022, https://www.splunk.com/en_us/newsroom/press-releases/2022/state-of-security-2022-report-reveals-increase-in-cyberattacks-while-security-talent-remains-scarce.html.

[2] R. Barber, "Hacking techniques: the tools that hackers use, and how they are evolving to become more sophisticated," *Computer Fraud & Security*, vol. 3, pp. 9–12, 2001.

[3] R. Islam, M. O. F. Rokon, A. Darki, and M. Faloutsos, "Hackerscope: the dynamics of a massive hacker online ecosystem," *Social Network Analysis and Mining*, vol. 11, no. 1, pp. 56–12, 2021.

[4] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl, "Security developer studies with GitHub users: exploring a convenience sample," in *Proceedings of the Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pp. 81–95, Santa Clara CA USA, July 2017.

[5] R. Kaksonen, T. Järvenpää, J. Pajukangas, M. Mahalean, and J. Röning, "100 popular open-source infosec tools," in *Proceedings of the IFIP International Conference on ICT Systems Security and Privacy Protection*, pp. 181–195, Springer, Oslo, Norway, June, 2021.

[6] H. Perl, S. Dechand, M. Smith et al., "Vccfinder: finding potential vulnerabilities in open-source projects to assist code audits," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 426–437, Denver Colorado, USA, October, 2015.

[7] M. Meli, M. R. McNiece, and B. Reaves, "How bad can it git? characterizing secret leakage in public github repositories," in *Proceedings of the NDSS Symposium 2023*, NDSS, San Diego, CA, USA, March 2019.

[8] M. O. F. Rokon, R. Islam, A. Darki, E. E. Papalexakis, and M. Faloutsos, "SourceFinder: finding malware Source-Code from publicly available repositories in GitHub," in *Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pp. 149–163, San Sebastian, Spain, October, 2020.

[9] AlphaSeclab, "awesome-security-collection," https://github.com/alphaSeclab/awesome-security-collection.

[10] Kali, "Kali tools," https://www.kali.org/tools/all-tools/.

[11] Y. Guo, Z. Liu, C. Huang et al., "Cyberrel: joint entity and relation extraction for cybersecurity concepts," in *Proceedings of the International Conference on Information and Communications Security*, pp. 447–463, Springer, Chongqing, China, November 2021.

[12] I. Sarhan and M. Spruit, "Open-cykg: an open cyber threat intelligence knowledge graph," *Knowledge-Based Systems*, vol. 233, Article ID 107524, 2021.

[13] Z. Li, J. Zeng, Y. Chen, and Z. Liang, "Attackg: constructing technique knowledge graph from cyber threat intelligence reports," in *Proceedings of the Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security*, pp. 589–609, Copenhagen, Denmark, September 2022.

[14] Y. Ren, Y. Xiao, Y. Zhou, Z. Zhang, and Z. Tian, "Cskg4apt: a cybersecurity knowledge graph for advanced persistent threat organization attribution," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, pp. 1–15, 2022.

[15] J. Undercoffer, A. Joshi, and J. Pinkston, "Modeling computer attacks: an ontology for intrusion detection," in *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, pp. 113–135, Springer, Pittsburgh, PA, USA, September 2003.

[16] M. Iannacone, S. Bohn, G. Nakamura et al., "Developing an ontology for cyber security knowledge graphs," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, pp. 1–4, Oak Ridge TN USA, April 2015.

[17] Z. Syed, A. Padia, T. Finin, L. Mathews, and A. Joshi, "Uco: a unified cybersecurity ontology," in *Proceedings of the*

Workshops at the thirtieth AAAI conference on artificial intelligence, Phoenix Arizona, February 2016.

[18] A. Pingle, A. Piplai, S. Mittal, A. Joshi, J. Holt, and R. Zak, "Relext: relation extraction using deep learning approaches for cybersecurity knowledge graph improvement," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 879–886, Vancouver British Columbia Canada, August 2019.

[19] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Findings from github: methods, datasets and limitations," in *Proceedings of the 2016 ieee/acm 13th working conference on mining software repositories (msr)*, IEEE, Austin, TX, USA, May 2016.

[20] T. Unruh, B. Shastry, M. Skoruppa et al., "Leveraging flawed tutorials for seeding Large-Scale web vulnerability discovery," in *Proceedings of the 11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, USA, June 2017.

[21] Y. Qian, Y. Zhang, N. Chawla, Y. Ye, and C. Zhang, "Malicious repositories detection with adversarial heterogeneous graph contrastive learning," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 1645–1654, Atlanta GA USA, October 2022.

[22] D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, "Committed to trust: a qualitative study on security & trust in open source software projects," in *Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1880–1896, IEEE, San Francisco, CA, USA, May 2022.

[23] X. Zhang, T. Wang, G. Yin, C. Yang, Y. Yu, and H. Wang, "Devrec: a developer recommendation system for open source repositories," in *Proceedings of the International Conference on Software Reuse*, pp. 3–11, Springer, Salvador, Brazil, May 2017.

[24] Y. Wan, L. Chen, G. Xu, Z. Zhao, J. Tang, and J. Wu, "Scsminer: mining social coding sites for software developer recommendation with relevance propagation," *World Wide Web*, vol. 21, no. 6, pp. 1523–1543, 2018.

[25] T. Zhou, W. Wang, and S. Zhao, "Open source galaxy: heterogeneous information networks in social coding," in *Proceedings of the 2021 IEEE 6th International Conference on Big Data Analytics (ICBDA)*, pp. 349–355, IEEE, Xiamen, China, March 2021.

[26] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, "Network attacks: taxonomy, tools and systems," *Journal of Network and Computer Applications*, vol. 40, pp. 307–324, 2014.

[27] C. Sas and A. Capiluppi, "Antipatterns in software classification taxonomies," *Journal of Systems and Software*, vol. 190, Article ID 111343, 2022.

[28] J. S. Eder, "Knowledge graph based search system," 2012, https://patents.google.com/patent/US20120158633A1/en.

[29] C. Peng, F. Xia, M. Naseriparsa, and F. Osborne, "Knowledge graphs: opportunities and challenges," *Artificial Intelligence Review*, pp. 1–32, 2023.

[30] Wikipedia, "Ontology definition from wikipedia," 2022, https://en.wikipedia.org/wiki/Ontology.

[31] T. Yadav and A. M. Rao, "Technical aspects of cyber kill chain," in *Proceedings of the International Symposium on Security in Computing and Communication*, pp. 438–452, Springer, Kochi, India, August 2015.

[32] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: design and philosophy,"Technical report, McLean, VI, USA.

[33] P. E. Kaloroumakis and M. J. Smith, "Toward a knowledge graph of cybersecurity countermeasures," Tech. rep., Technical report, Mitre, McLean, VI, USA, 2021.

[34] Y. Jia, Y. Qi, H. Shang, R. Jiang, and A. Li, "A practical approach to constructing a knowledge graph for cybersecurity," *Engineering*, vol. 4, no. 1, pp. 53–60, 2018.

[35] F. K. Kaiser, U. Dardik, A. Elitzur et al., "Attack hypotheses generation based on threat intelligence knowledge graph," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–17, 2023.

[36] D. Dessi, F. Osborne, D. Reforgiato Recupero, D. Buscaldi, and E. Motta, "Cs-kg: a large-scale knowledge graph of research entities and claims in computer science," in *Proceedings of the Semantic Web–ISWC 2022: 21st International Semantic Web Conference*, pp. 678–696, Springer, Virtual Event, October 2022.

[37] N. F. Noy and D. L. McGuinness, "Ontology development: a guide to creating your first ontology," https://protege.stanford.edu/publications/ontology_development/ontology101.pdf.

[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[39] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[40] Y. Chen, "Convolutional neural network for sentence classification," M.Sc. thesis, University of Waterloo, Waterloo, Canada, 2015.

[41] Y. Wu, M. Schuster, Z. Chen et al., "Google's neural machine translation system: bridging the gap between human and machine translation," 2016, https://arxiv.org/abs/1609.08144.

[42] M. Schuster and K. Nakajima, "Japanese and Korean voice search," in *Proceedings of the 2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5149–5152, IEEE, Kyoto, Japan, March 2012.

[43] J. Zhao, Q. Yan, J. Li, M. Shao, Z. He, and B. Li, "Timiner: automatically extracting and analyzing categorized cyber threat intelligence from social data," *Computers & Security*, vol. 95, Article ID 101867, 2020.

[44] E. Aghaei, X. Niu, W. Shadid, and E. Al-Shaer, "Securebert: a domain-specific language model for cybersecurity," in *Proceedings of the Security and Privacy in Communication Networks: 18th EAI International Conference, SecureComm 2022*, pp. 39–56, Springer, Virtual Event, October 2022.

[45] V. Orbinato, M. Barbaraci, R. Natella, and D. Cotroneo, "Automatic mapping of unstructured cyber threat intelligence: an experimental study:(practical experience report)," in *Proceedings of the 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pp. 181–192, IEEE, Charlotte, NC, USA, October 2022.

[46] Github, "Jackaduma, secbert," https://github.com/jackaduma/SecBERT.

[47] Youngja, "Cybersecurity-embeddings word2vec," https://ebiquity.umbc.edu/resource/html/id/379/Cybersecurity-embeddings.

[48] Y. Zhou, Y. Tang, M. Yi, C. Xi, and H. Lu, "Cti view: apt threat intelligence analysis system," *Security and Communication Networks*, vol. 2022, Article ID 9875199, 15 pages, 2022.

[49] H. Jo, Y. Lee, and S. Shin, "Vulcan: automatic extraction and analysis of cyber threat intelligence from unstructured text," *Computers & Security*, vol. 120, Article ID 102763, 2022.
[50] GitHub, "The 2022 state of the octoverse," 2022, https://octoverse.github.com.
[51] Statcounter, "Operating system market share worldwide," 2022, https://gs.statcounter.com/os-market-share.
[52] T. Alsop, "Share of the global server market by operating system in 2018 and 2019," 2019, https://www.statista.com/statistics/915085/global-server-share-by-os/.
[53] C.-L. Hwang and K. Yoon, "Methods for multiple attribute decision making," in *Multiple Attribute Decision Making*, pp. 58–191, Springer, Berlin, Germany, 1981.