A System for Automated Open-Source Threat Intelligence Gathering and Management

Peng Gao[1*], Xiaoyuan Liu[1*], Edward Choi[1], Bhavna Soman[2], Chinmaya Mishra[2], Kate Farris[2], Dawn Song[1]
[*]Equal Contribution
[1]University of California, Berkeley [2]Microsoft Corporation
{penggao,xiaoyuanliu,edwardc1028,dawnsong}@berkeley.edu, {Bhavna.Soman,Chinmaya.Mishra,Kate.Farris}@microsoft.com

ABSTRACT

To remain aware of the fast-evolving cyber threat landscape, open-source Cyber Threat Intelligence (OSCTI) has received growing attention from the community. Commonly, knowledge about threats is presented in a vast number of OSCTI reports. Despite the pressing need for high-quality OSCTI, existing OSCTI gathering and management platforms, however, have primarily focused on isolated, low-level Indicators of Compromise. On the other hand, higher-level concepts (e.g., adversary tactics, techniques, and procedures) and their relationships have been overlooked, which contain essential knowledge about threat behaviors that is critical to uncovering the complete threat scenario. To bridge the gap, we propose SecurityKG, a system for automated OSCTI gathering and management. SecurityKG collects OSCTI reports from various sources, uses a combination of AI and NLP techniques to extract high-fidelity knowledge about threat behaviors, and constructs a security knowledge graph. SecurityKG also provides a UI that supports various types of interactivity to facilitate knowledge graph exploration.

CCS CONCEPTS

• **Information systems** → **Graph-based database models**; • **Security and privacy**; • **Computing methodologies** → **Information extraction**;

KEYWORDS

Threat Intelligence; Security Knowledge Graph

ACM Reference Format:
Peng Gao[1*], Xiaoyuan Liu[1*], Edward Choi[1], Bhavna Soman[2], Chinmaya Mishra[2], Kate Farris[2], Dawn Song[1] . 2021. A System for Automated Open-Source Threat Intelligence Gathering and Management. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 18–27, 2021, Virtual Event , China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3448016.3452745

1 INTRODUCTION

Sophisticated cyber attacks have plagued many high-profile businesses [7]. To remain aware of the fast-evolving threat landscape and gain insights into the most dangerous threats, open-source Cyber Threat Intelligence (OSCTI) [21] has received growing attention from the community. Commonly, knowledge about threats is presented in a vast number of OSCTI reports in various forms (e.g., threat reports, security news and articles [4, 5]). Despite the pressing need for high-quality OSCTI, existing OSCTI gathering and management systems [1, 8, 9], however, have primarily focused on simple Indicators of Compromise (IOCs) [22], such as signatures of artifacts, malicious file/process names, IP addresses, and domain names. Though effective in capturing isolated, low-level IOCs, these platforms cannot capture higher-level behaviors such as adversary tactics, techniques, and procedures [2], which are tied to the attacker's goals and thus much harder to change. As the volume of OSCTI sources increases day-by-day, it becomes increasingly challenging to maneuver through and correlate the myriad of sources to gain useful insights. Towards this end, there is a pressing need for a new system that can harvest and manage high-fidelity threat intelligence in an automated, intelligent, and principled way.

There are several major challenges for building such a system. First, OSCTI reports come in diverse formats: some reports contain structured fields such as tables and lists, and some reports primarily consist of unstructured natural-language texts. The platform is expected to be capable of handling such diversity and extracting information. Second, besides IOCs, OSCTI reports contain various other entities that capture threat behaviors. The platform is expected to have a wide coverage of entity and relation types to comprehensively model the threats. Third, accurately extracting threat knowledge from unstructured OSCTI texts is non-trivial. This is due to the presence of massive nuances particular to the security context, such as special characters (e.g., dots, underscores) in IOCs. These nuances limit the performance of most NLP modules (e.g., sentence segmentation, tokenization). Besides, some learning-based information extraction approaches require large annotated training corpora, which is expensive to obtain manually. Thus, how to programmatically obtain annotations becomes another challenge.

To bridge the gap, we built SecurityKG (~ 9K lines of Python code), a system for automated OSCTI gathering and management. SecurityKG collects OSCTI reports from various sources, uses a combination of AI and NLP techniques to extract high-fidelity knowledge about threat behaviors as security-related entities and relations, constructs a *security knowledge graph* containing the entity-relation triplets, and updates the knowledge graph by continuously ingesting new data. Specifically, SecurityKG has the following key components: (1) a set of fast and robust crawlers for collecting OSCTI reports from 40+ major security websites; (2) a

SIGMOD '21, June 18–27, 2021, Virtual Event , China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452745

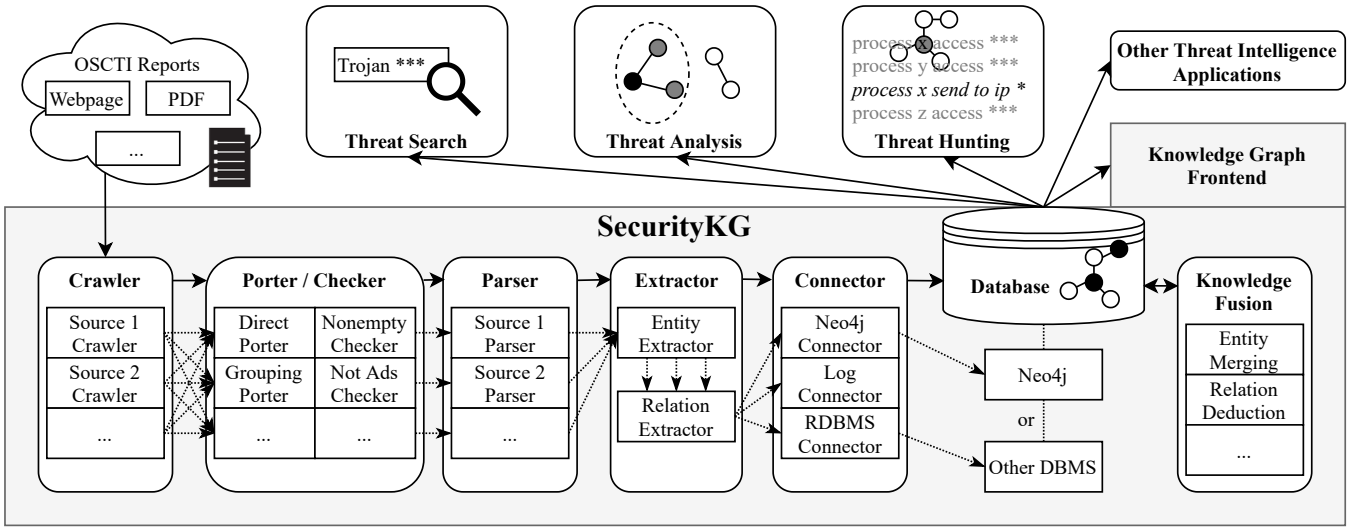

Figure 1: The architecture of SecurityKG. Arrows represent data flows.

security knowledge ontology that models a wide range of high-level and low-level security-related entities (e.g., IOCs, malware, threat actors, techniques, tools) and relations; (3) a combination of AI and NLP techniques (e.g., Conditional Random Fields [20]) to accurately extract entities and relations; specifically, we leverage data programming [27] to programatically create large training corpora; (4) an extensible backend system that manages all components for OSCTI gathering, knowledge extraction, and knowledge graph construction and persistence; (5) a UI that provides various types of interactivity to facilitate knowledge graph exploration.

Different from general knowledge graphs [10, 23, 25] that store and represent general knowledge (e.g., movies, actors), SecurityKG targets automated extraction and management of OSCTI knowledge for the security domain. SecurityKG is the first work in this space.

Demo video: https://youtu.be/8PDJSaTnLDc

2 SECURITYKG ARCHITECTURE

Figure 1 shows the architecture of SecurityKG. SecurityKG manages the lifecycle of security knowledge in four stages: collection (Crawler), processing (Porter/Checker, Parser, Extractor), storage (Connector, Database), and applications. In the collection stage, SecurityKG periodically and incrementally collects OSCTI reports from multiple sources. In the processing stage, SecurityKG parses the reports, extracts structured knowledge, and constructs a security knowledge graph based on a pre-defined ontology. In the storage stage, SecurityKG inserts the knowledge into backend databases for storage. Various applications (e.g., threat searching, threat analysis, threat hunting) can be built by accessing the security knowledge graph stored in the databases. SecurityKG also provides a frontend UI to facilitate knowledge graph exploration.

2.1 Backend System Design

To handle diverse OSCTI reports, the system needs to be scalable, and maintain a unified representation of all possible knowledge types in both known and future data sources. The system also needs to be extensible to incorporate new data sources and processing and storage components to serve the needs of different applications.

Scalability. To make the system scalable, we parallelize the processing procedure of OSCTI reports. We further pipeline the processing steps in the procedure to improve the throughput. Between different steps in the pipeline, we specify the formats of intermediate representations and make them serializable. With such pipeline design, we can have multiple computing instances for a single step and pass serialized intermediate results across the network, making multi-host deployment and load balancing possible.

Unified Knowledge Representation. To comprehensively represent security knowledge, we design an *intermediate CTI representation* and separate it from the security knowledge ontology. Intermediate CTI representation is a schema that covers relevant and potentially useful information in all data sources and lists out corresponding fields. We construct this schema by iterating through data sources, adding previously undefined types of knowledge, and merging similar fields. Specifically, our *source-dependent parsers* will first convert the original OSCTI reports into representations (i.e., Python objects in memory) that follow this schema by parsing the structured fields (e.g., fields identified by HTML tags). Then, our *source-independent extractors* will further refine the representations by extracting information (e.g., IOCs, malware names) from unstructured texts and putting it into the corresponding fields.

Directly using these intermediate representations results in inefficient storage. Furthermore, these long representations are not convenient for end users (e.g., threat analysts) to analyze. Thus, before merging them into the storage through connectors, SecurityKG refactors them to match our security knowledge ontology, which is separately designed and has clear and concise semantics.

Extensibility. To make the system extensible, we adopt a modular design, allowing multiple components with the same interface to work together in the same processing step. For example, SecurityKG by default uses a Neo4 connector to export knowledge into a Neo4j database [3]. However, if the user cares less about multi-hop relations, he/she may switch to a RDBMS using a SQL connector. Similarly, parsers and extractors can be switched or extended (sharing the same input/output formats), making the system extendable. Furthermore, the system can be configured through a

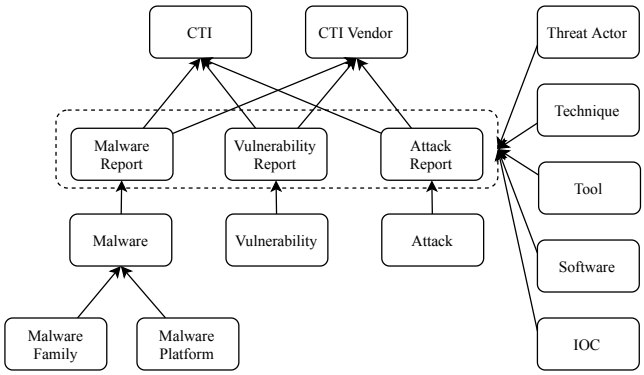

Figure 2: Security knowledge ontology

user-provided configuration file, which specifies the set of components to use and the additional parameters (e.g., threshold values for entity recognition) that are passed to these components.

2.2 OSCTI Reports Collection

We built a crawler framework that has 40+ crawlers for collecting OSCTI reports from major security sources (each crawler handles one data source), covering threat encyclopedias, blogs, security news, etc. The crawler framework schedules periodic execution and reboot after failure for different crawlers in an efficient and robust manner. It also has a multi-threaded design to boost the efficiency, achieving a throughput of approximately 350+ reports per minute on a single deployed host. In total, we have collected over 120K+ OSCTI reports and the number is still increasing.

2.3 Security Knowledge Ontology Design

Figure 2 shows our security knowledge ontology, which specifies the types of security-related entities and relations in the security knowledge graph. Based on our observations of OSCTI data sources, we categorize OSCTI reports into three types: malware reports, vulnerability reports, and attack reports. For each report, we associate it with an entity of the corresponding type. Besides, reports are created by specific CTI vendors, and often contain information concepts on threat actors, techniques, tools, software, and various types of IOCs (e.g., file name, file path, IP, URL, email, domain, registry, hashes). Thus, we create entities for these concepts as well. Entities have relationships between them (e.g., <MALWARE_A, DROP, FILE_A> specifies a "DROP" relation between a "MALWARE" entity and a "FILE" entity), as well as attributes in the form of key-value pairs. By constructing such an ontology, we can capture different types of security knowledge in the system. Compared to other cyber ontologies [6, 28], our ontology targets a larger set. Figure 3 shows an example knowledge subgraph that follows this ontology.

2.4 Security Knowledge Extraction

We describe the steps inside the processing stage that extract security knowledge from the collected OSCTI report files (e.g., HTML, PDF). The porters take the input report files and convert them into *intermediate report representations*; they group multi-page reports and add metadata like ids, sources, titles, and original file locations and timestamps. The checkers work as filters on the list of intermediate report representations; they screen out irrelevant reports like empty pages or ads by running condition checks. The parsers are source-dependent, taking the advantage of prior knowledge of the source website structure and extracting keys and values from report files. They convert the list of intermediate report representations into a list of *intermediate CTI representations* (Section 2.1). The extractors further refine these intermediate CTI representations by completing some of the fields using entity recognition and relation extraction. Since the intermediate CTI representation is a unified format, the extractors are source-independent.

Next, we describe the design of the extractors.

Security-Related Entity Recognition. We adopt a Conditional Random Field (CRF) [20] model to extract security-related entities in unstructured texts. Compared to general named entity recognition, we are faced with two unique challenges: (1) presence of massive nuances particular to the security context; (2) lack of large annotated training corpora. To address the first challenge, as these nuances mostly exist in IOCs, we use a method called *IOC protection* proposed in our other work [14], by replacing IOCs with meaningful words in natural language context (e.g., the word "something") and restoring them after the tokenization procedure. This way, we guarantee that the potential entities are complete tokens. To address the second challenge, we programmatically synthesize annotations using data programming [27]. Particularly, we create labeling functions based on our curated lists of entity names. For example, the lists of threat actors, techniques, and tools are constructed from MITRE ATT&CK [2]. To train the CRF model, we use features such as word lemmas, pos tags, and word embeddings [24]. Since our model has the ability to leverage token-level semantics, it can outperform a naive entity recognition solution that relies on regex rules, and generalize to entities that are not in the training set.

Security-Related Relation Extraction. To extract relations between security-related entities, since it is relatively hard to programmatically synthesize annotations for relations, we adopt an unsupervised approach. In particular, we leverage the dependency-parsing-based IOC relation extraction pipeline proposed in our other work [14], and extend it to support the extraction of relation verbs between entities recognized by our CRF model.

2.5 Security Knowledge Graph Construction

As a final step, SecurityKG inserts the processed results into the backend storage using connectors. The connector merges the intermediate CTI representations into the corresponding storage backend by refactoring them to match our security knowledge ontology, such that the previously constructed security knowledge graph can be augmented with new knowledge.

Since we store the knowledge extracted from a large number of reports in the same knowledge graph, one potential problem is that nodes constructed from different reports may refer to the same entity. We made the design choice that, in this step, we only merge nodes with exactly the same description text. It is possible that nodes with similar description texts actually refer to the same entity (e.g., same malware represented in different naming conventions by different CTI vendors). For these nodes, we merge them in a separate knowledge fusion stage, by creating a new node with unified attributes and migrating all relation edges. By separating

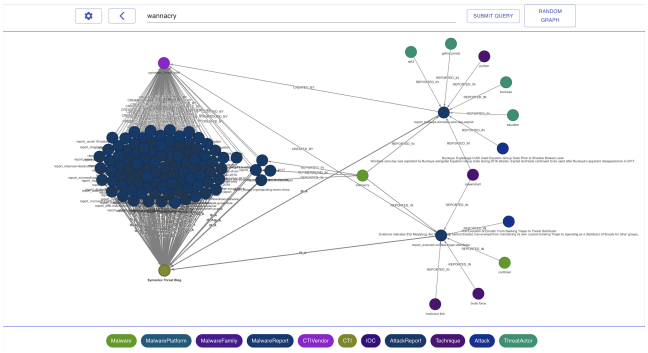

Figure 3: The web UI of SecurityKG

the knowledge fusion stage from the storage stage in the main pipeline, we can prevent early deletion of useful information.

2.6 Frontend UI Design

In order to facilitate knowledge graph exploration, we built a web UI using React and Elasticsearch. Figure 3 shows an example subgraph of security knowledge graph in our UI. Currently, the UI interacts with the Neo4j database through a Neo4j JS driver, and provides various functionalities to facilitate the exploration of the knowledge graph, which we describe next.

We built features to simplify user view. The user can zoom in and out and drag the canvas. Node names and edge types are displayed by default. Nodes are colored according to their types. When a node is hovered over, its detailed information will be displayed.

We built features that facilitate threat search and knowledge graph exploration. First, the UI provides multilingual query support so that the user can search information using keywords (through Elasticsearch) or Cypher queries (through Neo4j Cypher engine), which enables the user to easily identify targeted threats in the large graph. Second, the user can drag nodes around on the canvas. The UI actively responds to node movements to prevent overlap through an automatic graph layout using the Barnes-Hut algorithm [11], which calculates the nodes' approximated repulsive force based on their distribution. The dragged nodes will lock in place but are still draggable if selected. This node draggability feature helps the user define custom graph layouts. Third, the UI supports inter-graph navigation. When a node is double-clicked, if its neighboring nodes have not appeared in the view yet, these neighboring nodes will automatically spawn. On the contrary, once the user is done investigating a node, if its neighboring nodes or any downstream nodes are shown, double clicking on the node again will hide all its neighboring nodes and downstream nodes. This node expansion/collapse feature is essential for convenient graph exploration.

We built features that provide flexibility to the user. The user can configure the number of nodes displayed and the maximum number of neighboring nodes displayed for a node. The user can view the previous graphs displayed by clicking on the back button. The user can also fetch a random subgraph for exploration.

3 DEMONSTRATION OUTLINE

In our demo, we first show various usage scenarios of SecurityKG's UI. Specifically, we perform two keyword searches and one Cypher query search and demonstrate all supported features:

- *Keyword search for "wannacry"*: We first investigate the WannaCry ransomware by performing a keyword search. Throughout the investigation, we aim to demonstrate functionalities including detailed information display, node dragging, automatic graph layout, canvas zooming in/out, and node expansion/collapse. We will end the investigation with a subgraph that shows all the relevant information (entities) of the WannaCry ransomware.
- *Keyword search for "cozyduke"*: In the second scenario, we perform a keyword search of a threat actor, CozyDuke. We will investigate the techniques used by CozyDuke, and check if there are other threat actors that use the same set of techniques.
- *Cypher query search*: In the third scenario, we execute a specific Cypher query, `match(n) where n.name = "wannacry" return n`, to demonstrate that the same WannaCry node will be returned as in the first scenario. We then execute other queries.

Our demo video gives a walkthrough of these scenarios. In addition to threat search and knowledge graph exploration, we demonstrate the end-to-end automated data gathering and management procedure of SecurityKG. We will empty the database and apply SecurityKG to a number of OSCTI sources. We will demonstrate various system components, and provide insights into how OSCTI reports are collected, how entities and relations are extracted, and how information is merged into the knowledge graph so that the graph can continuously grow. The audience will have the option to try the UI and the whole system to gain deeper insights into various system components and the supported functionalities.

4 RELATED WORK

Besides existing OSCTI gathering and management systems [1, 8, 9], research progress has been made to better analyze OSCTI reports, including extracting IOCs [22], extracting threat action terms from semi-structured Symantec reports [19], understanding vulnerability reproducibility [26], and measuring threat intelligence quality [12, 21]. Research has also proposed to leverage individual OSCTI reports for threat hunting [14]. SecurityKG distinguishes from all these works in the sense that it targets automated construction of a knowledge graph particularly for the security domain, by extracting a wide range of security-related entities and relations from a large number of OSCTI reports using AI and NLP techniques.

In future work, we plan to connect SecurityKG with our query-based threat protection systems (e.g., attack investigation [17, 18], attack detection [15, 16], threat hunting [13, 14]) to enable knowledge-enhanced cyber threat protection.

5 CONCLUSION

We have presented SecurityKG, a system for automated OSCTI gathering and management. SecurityKG uses a combination of AI and NLP techniques to extract threat knowledge from a large number of collected OSCTI reports, and constructs a security knowledge graph to structuralize and persist the knowledge. SecurityKG has potential to empower a variety of security applications.

Acknowledgement. This work was supported by the 2020 Microsoft Security AI RFP Award, the Azure cloud computing platform, and the UC Berkeley Center for Long-Term Cybersecurity (CLTC).

REFERENCES

[1] [n.d.]. AlienVault OTX. https://otx.alienvault.com/.
[2] [n.d.]. MITRE ATT&CK. https://attack.mitre.org.
[3] [n.d.]. Neo4j. http://neo4j.com/.
[4] [n.d.]. PhishTank. https://www.phishtank.com/.
[5] [n.d.]. SecureList. https://securelist.com/.
[6] [n.d.]. Structured Threat Information eXpression. http://stixproject.github.io/.
[7] [n.d.]. The Equifax Data Breach. https://www.ftc.gov/equifax-data-breach.
[8] [n.d.]. ThreatCrowd. https://www.threatcrowd.org/.
[9] [n.d.]. ThreatMiner. https://www.threatminer.org/.
[10] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*.
[11] Josh Barnes and Piet Hut. 1986. A hierarchical O (N log N) force-calculation algorithm. *nature* (1986).
[12] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. 2019. Towards the detection of inconsistencies in public security vulnerability reports. In *USENIX Security*.
[13] Peng Gao, Fei Shao, Xiaoyuan Liu, Xusheng Xiao, Haoyuan Liu, Zheng Qin, Fengyuan Xu, Prateek Mittal, Sanjeev R Kulkarni, and Dawn Song. 2021. A system for efficiently hunting for cyber threats in computer systems using threat intelligence. In *ICDE*.
[14] Peng Gao, Fei Shao, Xiaoyuan Liu, Xusheng Xiao, Zheng Qin, Fengyuan Xu, Prateek Mittal, Sanjeev R Kulkarni, and Dawn Song. 2021. Enabling efficient cyber threat hunting with cyber threat intelligence. In *ICDE*.
[15] Peng Gao, Xusheng Xiao, Ding Li, Kangkook Jee, Haifeng Chen, Sanjeev R. Kulkarni, and Prateek Mittal. 2020. Querying streaming system monitoring data for enterprise system anomaly detection. In *ICDE*.
[16] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R. Kulkarni, and Prateek Mittal. 2018. SAQL: A stream-based query system for real-time abnormal system behavior detection. In *USENIX Security*.
[17] Peng Gao, Xusheng Xiao, Zhichun Li, Kangkook Jee, Fengyuan Xu, Sanjeev R. Kulkarni, and Prateek Mittal. 2019. A query system for efficiently investigating complex attack behaviors for enterprise security. In *VLDB*.
[18] Peng Gao, Xusheng Xiao, Zhichun Li, Fengyuan Xu, Sanjeev R. Kulkarni, and Prateek Mittal. 2018. AIQL: Enabling efficient attack investigation from system monitoring data. In *USENIX ATC*.
[19] Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. 2017. Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In *ACSAC*.
[20] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *ICML*.
[21] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. 2019. Reading the tea leaves: A comparative analysis of threat intelligence. In *USENIX Security*.
[22] Xiaojing Liao, Kan Yuan, XiaoFeng Wang, Zhou Li, Luyi Xing, and Raheem Beyah. 2016. Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *CCS*.
[23] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. 2013. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*.
[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*.
[25] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.
[26] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. 2018. Understanding the reproducibility of crowd-reported security vulnerabilities. In *USENIX Security*.
[27] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. In *NeurIPS*.
[28] Zareen Syed, Ankur Padia, Tim Finin, Lisa Mathews, and Anupam Joshi. 2016. UCO: A unified cybersecurity ontology. *UMBC Student Collection* (2016).