THREATKG: An AI-Powered System for Automated Open-Source Cyber Threat Intelligence Gathering and Management

Peng Gao
Virginia Tech
Blacksburg, VA, USA
penggao@vt.edu

Xiaoyuan Liu
University of California, Berkeley
Berkeley, CA, USA
xiaoyuanliu@berkeley.edu

Edward Choi
University of California, Berkeley
Berkeley, CA, USA
edwardc1028@berkeley.edu

Sibo Ma
University of California, Berkeley
Berkeley, CA, USA
siboma@berkeley.edu

Xinyu Yang
Virginia Tech
Blacksburg, VA, USA
xinyuyang@vt.edu

Dawn Song
University of California, Berkeley
Berkeley, CA, USA
dawnsong@berkeley.edu

Abstract

Open-source cyber threat intelligence (OSCTI) has become essential for keeping up with the rapidly changing threat landscape. However, current OSCTI gathering and management solutions mainly focus on structured Indicators of Compromise (IOC) feeds, which are low-level and isolated, providing only a narrow view of potential threats. Meanwhile, the extensive and interconnected knowledge found in the unstructured text of numerous OSCTI reports (e.g., security articles, threat reports) available publicly is still largely underexplored.

To bridge the gap, we propose THREATKG, an automated system for OSCTI gathering and management. THREATKG efficiently collects a large number of OSCTI reports from multiple sources, leverages specialized AI-based techniques to extract high-quality knowledge about various threat entities and their relationships, and constructs and continuously updates a threat knowledge graph by integrating new OSCTI data. THREATKG features a modular and extensible design, allowing for the addition of components to accommodate diverse OSCTI report structures and knowledge types. Our extensive evaluations demonstrate THREATKG's practical effectiveness in enhancing threat knowledge gathering and management.

CCS Concepts

• **Security and privacy** → *Vulnerability management*; Intrusion detection systems.

Keywords

threat intelligence; threat knowledge graph; security information extraction; deep learning

ACM Reference Format:
Peng Gao, Xiaoyuan Liu, Edward Choi, Sibo Ma, Xinyu Yang, and Dawn Song. 2024. THREATKG: An AI-Powered System for Automated Open-Source Cyber Threat Intelligence Gathering and Management. In *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis (LAMPS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3689217.3690613

The first two authors contributed equally to this research.

LAMPS '24, October 14–18, 2024, Salt Lake City, UT, USA

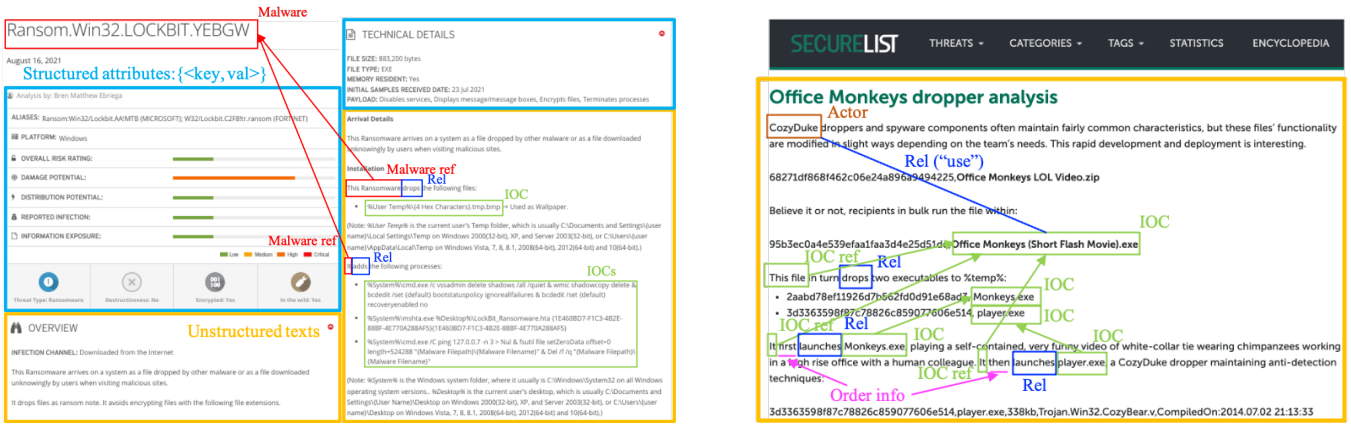
ACM ISBN 979-8-4007-1209-8/24/10
https://doi.org/10.1145/3689217.3690613

1 Introduction

Despite the dramatic growth in expenses on operational security, we are still witnessing numerous targeted cyber attacks. These sophisticated attacks leverage various types of exploits and vulnerabilities to penetrate into the system and steal valuable data. Many high-profile businesses were plagued with huge losses [3, 6]. To counter these attacks, it is crucial to always remain aware of the fast-evolving cyber threat landscape and gain up-to-date knowledge about the dangerous threats. For this reason, security researchers and practitioners actively gather and summarize knowledge about cyber threats from past incidents and share the knowledge to the public. Providing a form of evidence-based knowledge, such open-source cyber threat intelligence (OSCTI) [37] has the potential to empower various downstream defensive solutions and has received wide attention.

However, existing OSCTI gathering and management solutions are inadequate for the increasing complexity and diversity of cyber threats. They mainly focus on collecting and disseminating *structured* Indicator of Compromise (IOC) feeds [38]. IOCs are forensic artifacts of intrusions such as hashes of malware samples, names of malicious files/processes, and IP addresses and domains of command-and-control (C&C) servers. Some examples of platforms that share IOCs are PhishTank [19] and OpenPhish [18] for phishing URLs and Abuse.ch [9] for malware names and hashes. However, these low-level and disconnected IOCs are unable to reveal the complete threat scenario, such as how the threat unfolds into multiple steps, which is common in most sophisticated attacks nowadays [30]. Defensive solutions that rely on these IOCs are easy to evade when the attacker changes her tools and their signatures [37].

In contrast, ***a large number of unstructured OSCTI reports have been overlooked***, These reports are composed and shared by security researchers and practitioners on public websites to summarize threat behaviors in *natural language text*. Some examples of OSCTI reports are threat encyclopedia pages [14, 24], security articles and blogs [5, 21], security news [23], etc. These reports contain not only IOCs, but also other types of *threat knowledge entities*, such as threat actors, adversary tactics, techniques, and procedures (TTPs). Moreover, these reports contain the *semantic relationships between entities* that indicate their interactions (e.g., the read relationship between two IOCs /bin/tar and /etc/passwd implies the attacker



(a) Semi-structured threat encyclopedia report **(b) Unstructured security article**

Fig. 1: Example OSCTI reports that contain rich threat knowledge. (a) Semi-structured report snippet [7] from the Trend Micro threat encyclopedia. The report describes the ransomware, `Ransom.Win32.LOCKBIT.YEBGW`. (b) Unstructured report snippet [4] from the Securelist blog. The report describes the `CozyDuke` threat actor.

gathers user credentials into an archive file to be sent out). These relationships provide a form of connected knowledge with more context about cyber threats, which is critical to uncovering multi-step threat behaviors. Studies have shown that defenses based on such connected knowledge (e.g., IOC interactions [30]) are more robust, as they capture the threat behaviors that are aligned to the adversary goals and are harder to change. However, existing solutions lack the ability to automatically extract such comprehensive knowledge from natural language OSCTI text.

Goal and challenges. We aim to design and build a new system for gathering and managing OSCTI that can (1) automatically extract high-quality knowledge from a large number of unstructured OSCTI reports from various sources, and (2) store and organize such knowledge in a unified knowledge base that can provide comprehensive views of different threats. The key challenge is three-fold:

(1) *Unified knowledge representation:* To model the threats comprehensively, the system needs to cover a wide range of entity and relation types. Moreover, OSCTI reports collected from different sources have heterogeneous formats: some reports have structured fields and some reports are mainly composed of text, as shown in Fig. 1. Also, not all reports from a source are relevant to threats; some of them may be about advertisements or product promotions, as reported in [38]. Therefore, the system needs to handle such diversity, filter out irrelevant reports, and unify the gathered knowledge.

(2) *Accurate knowledge extraction:* Accurately extracting threat knowledge from natural language text is a challenging task. This is because of the presence of many nuances specific to the security context, such as special characters (e.g., dots, underscores) in IOCs. These nuances limit the performance of most off-the-shelf natural language processing (NLP) tools for information extraction. Moreover, collecting large annotated corpora is critical for training knowledge extraction models. A key challenge for the threat knowledge extraction domain is the lack of labeled datasets that encompass the diverse range of entity and relation types that we focus on.

(3) *Efficient knowledge management:* New OSCTI reports are being released every day that contain fresh threat knowledge. The system needs to continuously collect the latest reports from multiple OSCTI sources, gather new knowledge, and integrate the knowledge to update its knowledge base. The system also needs to be extensible to incorporate new OSCTI sources and report formats.

Contributions. We propose THREATKG (∼26K lines of code), a system for automated OSCTI gathering and management. THREATKG collects a large number of OSCTI reports from various sources, filters out irrelevant reports and extracts high-fidelity threat knowledge using AI-based techniques, constructs a *threat knowledge graph*, and updates the knowledge graph by continuously ingesting new data.

To model the threats comprehensively, THREATKG employs a hierarchical threat knowledge ontology that covers a wide range of entities. The relations between these entities provide information about both detailed threat behaviors and high-level threat contexts. To handle diverse OSCTI report formats and generalize well to new formats and knowledge, THREATKG separates the knowledge extraction process into source-dependent parsing and source-independent extraction. To deeply understand the semantic meaning and connections between targeted entities, THREATKG employs specialized deep learning-based techniques to handle the nuances and accurately extract the knowledge. We further leverage data programming techniques [43] to programmatically build synthetic annotations to train these models. THREATKG employs an *extensible system architecture* to continuously gather new knowledge in a timely manner. The architecture coordinates all individual components in a modular way, enabling efficient parallelization. Existing components can be turned off or updated, and new components can be easily added following the common interface. This allows THREATKG to incorporate new OSCTI sources or knowledge types.

To demonstrate the use cases of the threat knowledge graph, we develop two applications: (1) a graphical user interface application for visualizing, exploring, and searching the knowledge graph (demo video [26]); (2) a question answering system to facilitate threat knowledge acquisition using natural language (demo video [27]).

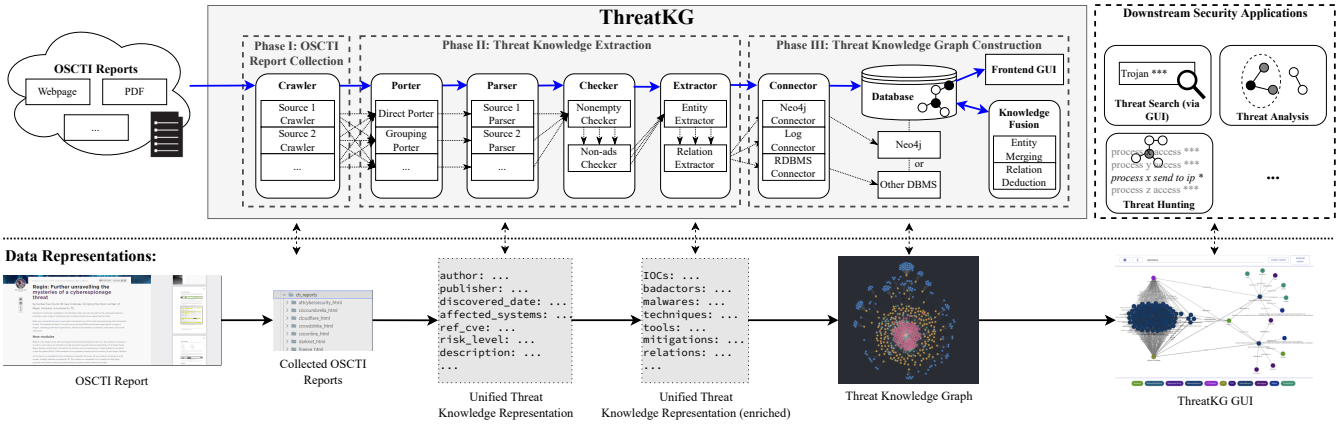


Fig. 2: Architecture of THREATKG. Arrows between system components indicate data flows.

2 System Overview

Fig. 2 illustrates the architecture of THREATKG, which consists of three phases: (1) OSCTI report collection, (2) threat knowledge extraction, and (3) threat knowledge graph construction. Each phase consists of one or several processing steps (e.g., Parser, Extractor). In Phase I, THREATKG collects OSCTI reports from a wide range of sources (Crawler). In Phase II, THREATKG aggregates multi-page report files (Porter), parses the reports (Parser), filters out non-threat reports (Checker), and extracts threat knowledge (Extractor). In Phase III, THREATKG constructs a TKG and stores it in the database. THREATKG is fully automated. It collects new reports periodically and incrementally, and extracts new knowledge from them. It then integrates the new knowledge into the threat knowledge graph.

Fig. 1a shows a report snippet from the Trend Micro threat encyclopedia [7] that describes the Ransom.Win32.LOCKBIT.YEBGW ransomware. The report has a semi-structured format, with some structured fields that provide attributes of the malware (e.g., aliases and platform) and some natural language text that provides detailed behaviors (e.g., dropping a file). Fig. 1b shows a report snippet from the Securelist blog [4] that describes the Office Monkeys dropper used by the CozyDuke threat actor. It primarily contains natural language text. We can observe that OSCTI reports have diverse formats and contain rich threat knowledge. We annotated representative entities and relations in the report snippets. Some entity-relation triplets reveal specific threat behavior steps, such as <Office Monkeys (Short Flash Movie).exe, launch, player.exe>. The text may also indicate the sequential order of some steps, such as "...first...then..." in Fig. 1b. Some triplets provide high-level threat contexts, such as the CozyDuke actor uses the Office Monkeys (Short Flash Movie).exe dropper file to perform the attack. These relations may not be explicitly expressed by words in the text. We take care of the extration of the temporal order using dependency parsing and the relations that are not explicitly expressed by words using neural relation extraction.

3 Report Collection & Ontology

3.1 OSCTI Report Crawlers

We have developed a robust multi-threaded crawler framework that manages crawlers to collect OSCTI reports from various security websites, given in Table VIII (Appendix). These websites include threat encyclopedias [14, 24], enterprise security blogs [5, 21], influential personal security blogs [20], security news [23], etc. They provide a rich source of threat knowledge, covering different types of threats such as malware, vulnerabilities, and attack campaigns.

Our crawler framework can handle the specific layout structure of each website and collect report URLs for fetching the content. It can deal with both static pages and dynamically generated content (e.g., "View More" in [5]). The framework schedules periodic execution and reboot after failure for each crawler, ensuring robustness and reliability. To improve the crawling efficiency, the framework employs a multi-threaded design that allows parallel execution of multiple crawlers, as well as fetching multiple reports for each crawler. With THREATKG's extensible architecture, new OSCTI sources can be easily added by adding a corresponding crawler and a parser.

To expand the knowledge coverage, we additionally collected OSCTI reports from APTnotes [11], a repository of publicly-available reports related to malicious campaigns/activities/software that have been associated with vendor-defined APT groups. These reports are in PDF format and are typically longer and more detailed than the security webpages, which provide complementary threat knowledge.

These OSCTI sources provide different kinds of threat knowledge, which we categorize into three broad types: malware reports, vulnerability reports, and attack reports.

- *Malware reports* and *vulnerability reports* are semi-structured reports that contain knowledge about malware or vulnerabilities. They are collected from threat encyclopedias, such as [14, 24]. These reports usually have a title indicating the name of the malware/vulnerability entity, followed by structured fields indicating the attributes of the entity and a natural language description of the behaviors of the entity. Fig. 1a shows an example malware report snippet on the Ransom.Win32.LOCKBIT.YEBGW ransomware.
- *Attack reports* are unstructured reports that contain knowledge about attack campaigns. They are collected from security blogs and news, such as [5, 20, 21, 23]. These reports mainly contain natural language text describing the context and behaviors of attack campaigns. Fig. 1b shows an example attack report snippet on the CozyDuke APT attack. CozyDuke is the name of the threat actor/group that is responsible for the attack.

3.2 Hierarchical Threat Knowledge Ontology

To model the threats comprehensively, we enumerate key knowledge pieces in our collected reports and design a hierarchical threat knowledge ontology. Our ontology (shown in Fig. 3) consists of three layers and covers various entities and relations for both low-level threat behaviors and high-level threat contexts.

The report context layer contains report-level knowledge. We create an entity for each report and associate it with attributes such as title, URL, publication date, etc. These entities help threat analysts connect other entities (e.g., malware, IOCs, TTPs) from the same report and form a comprehensive view of the threat. Threat analysts can also follow the URL attribute to view the original report and obtain more context. Moreover, we create entities for the specific authors and CTI vendors who write and create the reports.

The threat behavior layer contains knowledge about low-level threat behaviors, which are represented by IOCs and their relations. Previous studies [30, 44] have shown that these relations reveal how the threat progresses through connected steps. This knowledge can help identify system call events (e.g., process reading a file) that belong to the attack sequence. For example, in Fig. 1b, two filename IOCs, Office Monkeys (Short Flash Movie).exe and player.exe, have a launch relation. Therefore, we consider various types of IOCs and their interaction verbs (e.g., read, write, open, send) as their relations in the threat behavior layer. Example IOC types are filename, filepath, IP, URL, domain, registry, and MD5/SHA1/SHA256 hashes.

The threat context layer provides high-level contexts essential for a comprehensive understanding. This layer includes various entities, including: (1) malware, (2) vulnerabilities, (3) threat actors (e.g., CozyDuke APT actor [4]), (4) tactics and techniques (e.g., spearphishing link [16]), (5) vulnerable software (e.g., Microsoft Word), (6) security-related tools (e.g., Mimikatz), and (7) mitigations (e.g., data backup), These entities have different types of relations among them. We use TYPE_ENT to denote an entity placeholder of the type "TYPE". Some examples of entity-relation triplets in this layer are <ACTOR_ENT, use, MALWARE_ENT>, <ACTOR_ENT, use, TOOL_ENT>, and <SOFTWARE_ENT, has, VULNERABILITY_ENT>.

Entities in different layers can be related. For example, entities in the threat behavior layer and the threat context layer that are extracted from the same report are related to the report entity through a reported_in relation. In Fig. 1a, the malware entity Ransom.Win32.LOCKBIT.YEBGW is related to several filepath IOC entities through an add relation (after we perform coreference resolution). In Fig. 1b, the threat actor entity CozyDuke is related to the filename IOC entity Office Monkeys (Short Flash Movie).exe through a use relation. Entities can also have attributes in the form of key-value pairs (e.g., type of a malware, version of a vulnerable software). The three layers of ontology collectively model the threats from multiple dimensions and in different granularities.

4 Threat Knowledge Extraction

4.1 Report Parsing and Relevance Checking

The crawlers collect the OSCTI reports and the porters aggregate them into multi-page files. Each OSCTI source has a specific structure, so we use different parsers to parse them (i.e., parsers are source-dependent). The parsers convert each report into a unified

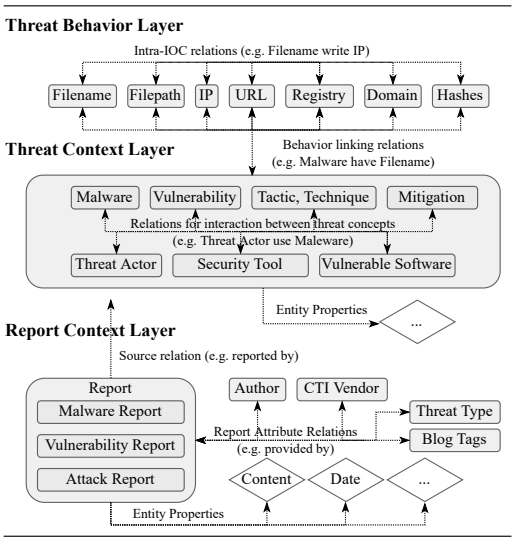


Fig. 3: Hierarchical threat knowledge ontology

threat knowledge representation (UTKR), which is a JSON schema. We create this schema by iterating through OSCTI reports and adding fields for new types of knowledge. It has fields such as title, author, and placeholders for entities and relations. The parsers also parse the unstructured text blocks and put them into the UTKRs. The extractors then enrich the UTKRs by extracting more entities and relations from the unstructured text. Having a unified representation increases the system's modularity and extensibility; new components can be easily added as long as they work with the same schema design.

There could be irrelevant reports that do not contribute knowledge to model cyber threats, such as empty pages, advertisements, and product promotions. To filter out these reports, THREATKG employs a set of checkers that operate on the UTKRs produced by the parsers. Empty web pages can be easily filtered out. For ads and other irrelevant reports, we model the relevance checking process as a binary classification task and construct AI-based checkers. We engineer a set of useful features, including: (1) Keyword count and density: We count the number and proportion of keywords in the report title and body. We use a list of keywords from MITRE ATT&CK [16], such as threat actors, malware, tools, techniques, etc.; (2) IOC count and density: We extract IOCs using regex rules [13]. We only consider the report body, as most of the titles do not contain IOCs; (3) Report length: We measure the number of words in the report. We observe that a longer report is more likely to contain threat behaviors; (4) TF-IDF values: We calculate the TF-IDF (term frequency–inverse document frequency) value for each token in the report to prioritize frequent, unique tokens. We train various machine learning models (e.g., SVM, Random Forest, XGBoost, LightGBM) using these features and compare their performance in Section 7.1.

4.2 Threat Entity Extraction

The extractors are source-independent; every extractor extracts the targeted knowledge from the text in all reports, and the extraction does not depend on the specific layout structure of each source. By decoupling the knowledge extraction process into source-dependent parsing and source-independent extraction, THREATKG can easily

incorporate new OSCTI sources (via adding crawlers and parsers) and new knowledge entities and relations (via adding extractors).

For IOCs, we construct a set of regex rules [13] that cover a wide range of IOC types. ThreatKG incorporates these rules in a rule-based IOC extractor. For other types of entities that are hard to define using rules, ThreatKG employs a deep learning-based extractor to perform neural named entity recognition (NER). NER is an information extraction task that aims to identify and categorize named entities in text into pre-defined classes. Deep learning-based approaches have an advantage over conventional methods like Hidden Markov Model, as they do not require manual feature engineering and can better capture the semantic meaning and hidden patterns of text, resulting in more accurate extraction.

Compared to general text, OSCTI text has many nuances that are specific to the security context, such as dots, underscores, spaces, and slashes in IOCs. These nuances can cause errors in most basic NLP modules (e.g., sentence segmentation, tokenization), and affect the extraction techniques that rely on these modules. To deal with these nuances, we substitute the IOCs with meaningful words that fit the natural language context (e.g., word "FILE" for a file IOC token) before applying neural NER, and replace them with the original IOCs after extracting other entities.

To perform neural NER on OSCTI text, we construct a Bidirectional LSTM-CRF (BiLSTM-CRF) model [35] for our deep learning-based entity extractor. (1) First, we tokenize each input sentence and convert each token into an embedding vector using one-hot encoding. (2) Then, we feed the embeddings to the bidirectional LSTM (BiLSTM) layer, which has two LSTM networks that process the input sentence from both directions. LSTM (Long-Short Term Memory) [32] is known for its capability to capture long-range dependencies of tokens. However, a single LSTM can only access information from the past context. For tasks like NER, it is important to understand the context of a token from both past and future contexts. Therefore, we use another LSTM, which processes the information in a reverse order. The BiLSTM acts as a deep feature extractor that captures the sequential relationships among the input tokens. (3) The outputs from the BiLSTM are passed to a linear layer, which maps the features extracted by the BiLSTM from the feature space to the tag space. After mapping, the outputs are passed to a Conditional Random Field (CRF) layer, which predicts the optimal, joint tags for the whole sentence.

To prepare the training corpus, we use the BIO format to label tokens with entity tags. The BIO format uses three types of prefixes: (i) B- prefix, for a token at the start of an entity chunk, (ii) I- prefix, for a token within a chunk, and (iii) O- prefix, for a token outside a chunk. Some examples of tags are B-BADACTOR (for threat actor), B-MALWARE, B-TOOL, B-TECHNIQUE, B-MITIGATION, etc.

4.3 Threat Relation Extraction

Dependency parsing-based relation extraction. As discussed in Section 2, some relations are directly *associated with verbs* that describe the interaction between two entities (e.g., the drop relation between the malware entity and an IOC in Fig. 1a, the launch relation between the IOCs Office Monkeys (Short Flash Movie).exe and player.exe in Fig. 1b). Some other relations are *not explicitly* indicated by any words in the text (e.g., the use relation between CozyDuke and Office Monkeys (Short Flash Movie).exe in Fig. 1b).

For the first type, we design a dependency parsing-based relation extractor to identify the verbs that express the interaction between two entities. We adopt dependency parsing [34] to analyze the grammatical structure of a sentence and constructs a dependency tree. Then, we use a set of dependency grammar rules [30] to extract the subject-verb-object relations between the extracted entities. We also extract the *temporal order* of the interaction steps by looking for specific tokens (e.g., "first", "then"), if present.

Neural relation extraction. For the second type, the dependency parsing-based approach will not work, as these relations do not have explicit verbs in the text. Instead, we model the relation extraction as a *multi-class classification task*: given a sentence that contains two entities recognized by our entity extractors, we determine the relation class between them. The entities include the IOCs recognized by our IOC rules and the other entities recognized by our BiLSTM-CRF model. Some examples of relation classes are USE (i.e., using something to achieve a goal), CREATE (i.e., creating or making something that did not exist before), BREAK (i.e., stopping or preventing something from happening), FIND (i.e., finding or locating something), and ALIAS (i.e., two entities being synonyms). A complete list of relation classes in given in Table IX (Appendix). In general, two entities could have a relation when they co-occur within a certain distance. These entities could co-occur in the same sentence or in different sentences. In our current implementation, we focus on entities that co-occur in the same sentence, as they are more likely to produce high-quality relations based on our observations.

To perform neural relation extraction (RE) on OSCTI text, we construct a Piecewise Convolutional Neural Networks model with selective attention mechanism (PCNN-ATT) for our deep learning-based relation extractor. The Piecewise Convolutional Neural Networks (PCNN) model [47] is a variation of the Convolutional Neural Networks (CNN) model that is widely used for image and text classification tasks. However, PCNN is specially designed for relation extraction: it splits a sentence into three parts by the two entities and applies piecewise max pooling to each part, instead of using a single max pooling to merge features as in CNN. This way, PCNN can capture the structural information about the sentence and the two entities, and identify the important tokens between them that indicate the relation. We convert the sentences into embedding vectors using word embeddings from GoogleNews-vectors-negative300 [1], position embeddings, and part-of-speech embeddings (indicating the roles of the words). Moreover, we use an attention laye on top of the PCNN output to make the model focus on the tokens that are more relevant for relation extraction.

Before extracting relations, ThreatKG performs coreference resolution [36] to find all the expressions (e.g., pronouns) in the text that refer to the same entity. Fig. 1 shows some examples of entity coreferences indicated by the arrows. This way, the relation extractor can use the information from the resolved entities and the extracted triplets can form a comprehensive view of threat knowledge.

4.4 Data Programming

To train deep learning-based models for NER and RE, we need a large annotated corpus. However, manually annotating such a corpus

is costly: for NER, we need to tag each token in the text with a label in the BIO format; for RE, we need to label each sentence in the text with a relation class and the types and location spans of the entities in the sentence. Unlike other information extraction domains that have plenty of labeled datasets, there are no large annotated corpora for the threat knowledge extraction domain. To reduce the cost of obtaining supervision, we leverage data programming [43], which synthesizes annotations *programmatically* using unsupervised modeling of sources of weak supervision. Specifically, data programming obtains the domain knowledge expressed by subject matter experts through labeling functions (which could be noisy rules based on heuristics), and then denoises and integrates these sources of weak supervision to synthesize annotations. We use Snorkel [22], an open-source implementation of data programming, to programmatically create large training sets for our NER and RE tasks. Snorkel does not need any labeled data for training (i.e., unsupervised): after we construct labeling functions, it automatically learns and assigns weights to the labeling functions and produces a single set of noise-aware confidence-weighted labels for the input samples.

The most important step in synthesizing good annotations is to define noisy but useful labeling functions, which we spent most of our efforts on. To synthesize annotations for the NER task, we create labeling functions based on our curated list of entity keywords. For example, we construct the list of threat actors, malware, techniques, and tools from MITRE ATT&CK [16]. To synthesize annotations for the RE task, we create labeling functions based on distant supervision and checking the entity types and keywords existence.

Distant supervision [40] is a technique that uses an existing knowledge base to generate training data. The idea is that if two entities have a fact in the knowledge base, we can label any sentence that contains them as a positive example for the relation that the fact represents. This way, we can create a large number of (noisy) labeled sentences. For example, Freebase contains the fact that Barack Obama and Michelle Obama are married. We use this fact to label any sentence that has "Barack Obama" and "Michelle Obama" as a positive example for our marriage relation. In our threat knowledge extraction task, we use MITRE ATT&CK, which is a manually curated knowledge base by security experts for cyber adversary behaviors and can be downloaded as a JSON file. For example, for a sentence that has a threat actor entity and a malware entity, if the two entities are in the MITRE ATT&CK and have the "use" relation type, we label the sentence with the USE relation class.

We also create labeling functions based on heuristic rules that assign relation labels based on the entity types and the presence of keywords. For example, for the ALIAS relation, we verify that the two entities belong to the same type and look for keywords such as "alias" or "aka". By leveraging data programming, we can generate a large amount of training data with low human effort.

5 Extensible System Architecture

Threat knowledge graph construction. After the extractors enrich the UTKRs, THREATKG constructs the threat knowledge graph from them and stores it in the database for persistence. Storing the UTKRs directly is inefficient and makes it hard for end users (e.g., threat analysts) to understand and analyze them. Therefore, THREATKG converts these intermediate representations to match the threat knowledge ontology, which has clear and concise semantics for entities and relations. This ontology is designed separately from the UTKRs. THREATKG then integrates the transformed representations into the database using its connectors. Currently, THREATKG uses Neo4j, the leading graph database, as its storage, where nodes are entities and edges are relations. Each node has a category (e.g., malware or threat actor), a unique name (e.g., specific malware name), and a set of attributes. THREATKG can easily support new database backends by adding the corresponding connectors without changing the previous components in its processing pipeline.

Modularity and extensibility. THREATKG adopts a modular design to make the system extensible, which enables multiple system components in the same processing step to have the same input/output interface. For example, THREATKG uses multiple crawlers to collect OSCTI reports from various sources, and different porters to import report data from various formats, such as HTML, PDF, and compressed files. Moreover, THREATKG supports rich configuration options: the system can be customized through a configuration file, which defines the components to use and the parameters to pass to them (e.g., threshold values for NER). With this design, existing components can be easily replaced or added.

We parallelize the system components for the processing steps (e.g., crawlers, parsers, checkers, extractors) to improve the system efficiency. We define the formats of intermediate representations (i.e., UTKRs) and make them serializable between different processing steps. These UTKRs are enriched as they pass through the pipeline.

Continuous updating. THREATKG is automated and continuously running to provide the latest threat knowledge in a timely manner. The threat knowledge graph is updated incrementally with new reports being collected and new knowledge being extracted and integrated. Different sources may use different identifiers for the same entity. For example, "ZQuest" and "Z-Quest" refer to the same adware. To ensure consistency, THREATKG combines knowledge from multiple sources using knowledge fusion: THREATKG scans all the entities and merges facts about the same entity by creating a new entity as the result and moving all relations. A key challenge is that entities with similar names may be different. For example, "Petya" and "NotPetya" are two ransomware with names satisfying a substring relation but are different entities. To address this challenge, THREATKG uses the contextual information stored with the entity and only merges two entities when they have a similar name (e.g., semantic similarity computed using word embeddings) above a threshold, no conflicts in their attributes, and operate in a similar environment (e.g., the same platform). By using contextual information and avoiding conflicts, THREATKG minimizes the information loss in its knowledge fusion, while providing a consistent and comprehensive view of entities from multiple sources.

6 Downstream Security Applications

Various security applications can be built upon the threat knowledge graph to enhance the defenses. In this section, we present two applications that we built to facilitate threat knowledge graph visualization and exploration and threat knowledge acquisition.

6.1 GUI for Threat Visualization and Exploration

To facilitate threat search and threat knowledge graph exploration, we built a GUI using React and Elasticsearch. The GUI interacts with the threat knowledge graph stored in the Neo4j database and provides various types of interactivity. As illustrated in our demo video [26], the user can zoom in/out and drag the canvas to adjust the view, click on a node or an edge to see the detailed information, and search information by keywords (using Elasticsearch) or Cypher queries (using Neo4j Cypher engine). Once the user drags a node, the GUI responds to the node movements to prevent overlap through an automatic graph layout using the Barnes-Hut algorithm [28], 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 feature helps the user create custom graph layouts for visualization.

The GUI also supports convenient threat knowledge graph navigation. The user can double-click on a node to expand or collapse its neighboring nodes. If the neighboring nodes are not in the view, they will appear when the node is double-clicked. If the neighboring nodes or any downstream nodes are in the view, they will disappear when the node is double-clicked again. The user can also adjust the number of nodes and the maximum number of neighboring nodes displayed for each node, and go back to the previous graphs displayed. Our GUI is not tied to the speci c database backend, and it can easily switch to a different database (e.g., RDF store) while providing the same functionalities.

6.2 Question Answering System

We built a QA system (named THREATQA) on top of TKG to facilitate threat knowledge acquisition. The user can ask a natural language question about the attributes or connections of a threat, and THREATQA will return the answer from the TKG. THREATQA can handle: (1) simple questions that ask for an entity's attribute or related entities, such as “Which CVE ID is exploited by EternalBlue?”, and (2) complex questions that require multi-hop reasoning, such as “What common techniques are used by DarkVishnya and Chimera?”

The system follows a three-stage pipeline:

Stage 1: Entity linking. To link the question to the TKG, THREATQA rst identi es the entities in the question using the neural NER approach described in Section 4.2. Then, for each entity, it searches for the most similar entity in the TKG among the entities of the same category. THREATQA links the entity in the question to the entity in the TKG with the highest similarity.

Stage 2: Question intent mapping. THREATQA uses a Roberta-based intent classi er [39] to identify the question's intent. Then, it nds the attribute or relation in the TKG that matches the question's intent. The attribute or relation helps THREATQA to locate a subgraph of the TKG that contains the answer.

Stage 3: Query synthesis and answer retrieval. THREATQA adopts a template-based approach to generate Cypher queries for different types of questions. It has carefully designed query templates (one example shown below) that encode the path between the entity in the TKG and the target answer. It lls in the linked entities into the query template that matches the asked attribute or relation. Then, it executes the query over the TKG stored in the Neo4j database and retrieves the nal answer. This approach ensures that the query is grammatically correct and reliable.

```
// Cypher query template
% // Identify techniques used by a threat actor, var1
% // Return: a list of techniques
MATCH(var1:Actor)-[rel:USE]->(var2:Technique)
WHEREACTOR_NAME IN var1.name
RETURNvar2.name
```

We built a GUI for THREATQA using React, as shown in our demo video [27]. The GUI displays the results of each QA processing stage: (1) the recognized question intent (“malware_type”), with the entities in the question and their categories; (2) the entity linking result (from the entity name “Downloader.Slime” in the question to the malware node in the TKG); (3) the synthesized Cypher query and the nal answer (“trojan house”). The GUI also allows the user to edit the Cypher query to investigate the malware further.

Table I: Statistics of our labeled ground-truth OSCTI dataset

Data Source	Category	# Reports
apt_notes	APT Reports	15
kaspersky_threat	Threat Encyclopedia	45
symantec_threat	Threat Encyclopedia	45
attcybersecurity	Enterprise Security Blog	12
crowdstrike	Enterprise Security Blog	6
securelist	Enterprise Security Blog	7
symantecthreatintelligence	Enterprise Security Blog	11
Total:		141

7 Evaluation

We built THREATKG (26K lines of code) upon several tools: Python for the system architecture, BeautifulSoup and Selenium for the crawlers, scikit-learn and Ray Tune (for hyperparameter optimization) for the checkers, PyTorch for the extractors, Snorkel for data programming, and Neo4j for the storage backend. We evaluate our system on several aspects, such as the accuracy of the knowledge extraction and the performance of the system. We aim to answer the following key research questions:

(RQ1) How well can THREATKG identify and lter out OSCTI reports that do not contain any cyber threat information?

(RQ2) How effectively can THREATKG extract threat knowledge from the OSCTI text? How much does the data programming technique enhance the extraction performance?

(RQ3) How does THREATKG compare with other baselines in extracting various types of threat knowledge?

(RQ4) For the runtime performance, is THREATKG ef cient enough to be practical for a real-world deployment?

Evaluation setup. We deployed THREATKG on a server with an AMD EPYC 7282 CPU (2.80GHz), an Nvidia GRID T4-16Q GPU with 16GB RAM, and Ubuntu 20.04 as the operating system. To evaluate the accuracy of THREATKG in extracting threat knowledge from OSCTI reports, we created a ground-truth labeled dataset from seven different OSCTI sources, namely: APTnotes attack reports, two threat encyclopedias, and four enterprise security blogs. These sources provide diverse and comprehensive OSCTI reports that cover various types of threat knowledge. We manually labeled 141 reports

Table II: Report checker performance (averaged for different classi ers)

	Symantec Threat Intelligence				Securelist				Webroot			
Training Procedure	Accuracy	F1	FPR	FNR	Accuracy	F1	FPR	FNR	Accuracy	F1	FPR	FNR
Source-speci c	93.33%	95.38%	21.21%	0.00%	81.14%	87.67%	53.62%	3.77%	78.33%	86.02%	64.10%	1.23%
Universal	94.29%	95.92%	13.64%	2.08%	80.04%	86.99%	54.35%	5.03%	73.75%	83.50%	76.92%	1.85%

Table III: Source-speci c checker results

	Symantec Threat Intelligence				Securelist				Webroot			
Models	Accuracy	F1	FPR	FNR	Accuracy	F1	FPR	FNR	Accuracy	F1	FPR	FNR
Logistic Regression	94.29%	96.00%	18.18%	0.00%	80.26%	87.18%	56.52%	3.77%	80.00%	87.10%	61.54%	0.00%
Random Forest	94.29%	96.00%	18.18%	0.00%	81.58%	88.33%	60.87%	0.00%	77.50%	85.71%	69.23%	0.00%
Linear SVM	94.29%	96.00%	18.18%	0.00%	80.26%	87.18%	56.52%	3.77%	80.00%	87.10%	61.54%	0.00%
Kernel SVM	88.57%	92.31%	36.36%	0.00%	82.89%	88.89%	52.17%	1.89%	80.00%	87.10%	61.54%	0.00%
LightGBM	94.29%	96.00%	18.18%	0.00%	82.89%	88.70%	47.83%	3.77%	77.50%	85.25%	61.54%	3.70%
XGBoost	94.29%	96.00%	18.18%	0.00%	78.95%	85.71%	47.83%	9.43%	75.00%	83.87%	69.23%	3.70%
Average	93.33%	95.38%	21.21%	0.00%	81.14%	87.67%	53.62%	3.77%	78.33%	86.02%	64.10%	1.23%

Table IV: Universal checker results

	Symantec Threat Intelligence				Securelist				Webroot			
Models	Accuracy	F1	FPR	FNR	Accuracy	F1	FPR	FNR	Accuracy	F1	FPR	FNR
Logistic Regression	94.29%	95.83%	9.09%	4.17%	84.21%	89.09%	34.78%	7.55%	82.50%	88.52%	53.85%	0.00%
Random Forest	94.29%	96.00%	18.18%	0.00%	76.32%	85.48%	78.26%	0.00%	70.00%	81.82%	92.31%	0.00%
Linear SVM	97.14%	97.96%	9.09%	0.00%	85.53%	90.43%	43.48%	1.89%	72.50%	83.08%	84.62%	0.00%
Kernel SVM	97.14%	97.96%	9.09%	0.00%	72.37%	83.20%	86.96%	1.89%	75.00%	84.37%	76.92%	0.00%
LightGBM	91.43%	93.88%	18.18%	4.17%	82.89%	88.29%	39.13%	7.55%	70.00%	81.25%	84.62%	3.70%
XGBoost	91.43%	93.88%	18.18%	4.17%	78.95%	85.45%	43.48%	11.32%	72.50%	81.97%	69.23%	7.41%
Average	94.29%	95.92%	13.64%	2.08%	80.04%	86.99%	54.35%	5.03%	73.75%	83.50%	76.92%	1.85%

from these sources according to the ontology we de ned. Table I summarizes the statistics of our ground-truth OSCTI dataset. For the entities, we used the BIO tagging scheme to mark their boundaries and types. For the relations, we labeled both the relation verbs (if any) and the relation classes between the entity pairs. We have 17 relation classes in total (shown in Table IX in Appendix), covering major types of threat behaviors. Two of our authors performed the labeling task independently and then cross-validated each other's results and resolved any disagreements.

7.1 RQ1: Accuracy of Irrelevant Report Filtering

To evaluate the checker performance, we created a dataset by randomly selecting 755 reports from three different OSCTI sources: Securelist [21], Symantec Threat Intelligence [5], and Webroot [25]. Out of these, 517 reports are relevant to cyber threats and 238 reports are irrelevant to cyber threats (e.g., reports about advertisements, security products, cybersecurity education).

OSCTI reports collected from different sources vary in their structures, writing styles, and topics. We wanted to examine how the distributional shift in the training data affects the performance of a classi er that predicts the relevance of a report to cyber threats. We conducted two experiments. In the rst experiment, we trained a source-speci c classi er for each source using only the data from that source. We then evaluated the classi er on the same source data. In the second experiment, we trained a universal classi er using the data from all sources combined. We then evaluated the classi er on each source data individually. We used six machine learning classi ers for both experiments: Logistic Regression, Random Forest, Linear SVM, SVM with RBF Kernel, XGBoost, and LightGBM. We split the data for each source into 70-10-20 for train/dev/test sets. We merged the train/dev/test sets from all sources to form the corresponding sets for the universal classi er.

Table II shows the results averaged for different models. Tables III and IV contain the results for each individual model. We observe: (1) For source-speci c classi ers, the average F1 scores are above 86% and the average false negative rates (FNRs) are below 3.77%. The false positive rates (FPRs) are higher. In our problem setting, a high FPR is acceptable as long as the FNR can be suf ciently low, because a high FNR means that many relevant reports (and the contained threat knowledge) are ltered out, while a high FPR just means that the system is conservative in ltering the reports. Note that our goal is to extract as much information as possible without overlooking threat-related articles. (2) The performance of the universal classi er does not bene t from more training data, and is worse than the source-speci c classi ers for some sources (e.g., Securelist and Webroot). This veri es the distributional shift problem in different OSCTI sources that we conjectured previously. Based on these observations, we recommend training classi ers for different sources separately to get better checker performance.

7.2 RQ2: Accuracy of Knowledge Extraction

Numerous works have successfully used BiLSTM-CRF to extract entities from natural language text, achieving excellent performance. In this RQ, we primarily focus on evaluating the performance of our relation extractor and the impact of data programming on this task.

Our labeled dataset shown in Table I contains 7308 relations. We randomly picked 16 reports and constructed the test set using the relations in them. In the remaining 125 reports, there are 1219

Table V: Relation extraction performance (aggregated)

	Precision	Recall	Accuracy	F1
W/O Data Programming	80%	78%	78%	79%
W/ Data Programming	85%	85%	85%	85%

"non-other" relations and 4615 OTHER relations (assigned when none of the other relation types match). This dataset is imbalanced and will negatively impact the trained model performance. Thus, we under-sampled the OTHER relations to make the dataset more balanced. After under-sampling, there are 1732 OTHER relations left. To expand the dataset, we manually labeled 805 additional "non-other" relations chosen from the same seven OSCTI sources. Finally, we created train/dev split of 87.5% and 12.5% respectively from the 2024 "non-other" relations and 1732 OTHER relations. The aggregated results for all relation classes are presented in Table V. The model achieved an F1 score 79%, which is reasonable considering the dif culty of the multi-class classi cation task (17 classes) and the limited size of the dataset (i.e., 2024 "non-other" relations that we labeled).

Effectiveness of data programming. To address the data scarcity issue, we applied data programming to generate more training instances. We labeled 2049 additional "non-other" relations and used all the 4615 OTHER relations from the 125 reports. We split the resulting dataset into train/dev subsets with a ratio of 87.5% and 12.5%, respectively. We used the same test set as before, which was manually labeled from 16 randomly selected reports. For correct evaluation setup, the test set only includes manual labels and does not include labels obtained via data programming.

The results in Table V show that data programming signi cantly improved the relation extraction performance, from 79% F1 to 85% F1. Moreover, the model performed better on the relation types that had fewer training instances in the previous experiment. For instance, the relation INJECT had an F1 score of 55% with 222 training instances in the previous experiment, but it increased to 72% with 558 training instances after data programming. These results con rm the effectiveness of data programming in creating more training data to enhance the model.

7.3 RQ3: Comparison with Existing Security Information Extraction Approaches

We compared THREATKG with two state-of-the-art security information extraction approaches, TTPDrill [33] and EXTRACTOR [44]., to further evaluate THREATKG 's effectiveness in extracting threat knowledge. We used the same 16 reports that we used for the test set in Section 7.2, which covered a wide range of threat scenarios, such as major OS platforms (Linux, Windows, IOS, and Android), well-known APT campaigns (Stuxnet and Beapy), and common types of cyber threats (malware and cryptojacking attacks). We ran TTPDrill and EXTRACTOR on these reports, and used the same THREATKG model that we trained in Section 7.2 for performance comparison.

Note that TTPDrill and EXTRACTOR are more limited than THREATKG in the scope of threat knowledge extraction. TTPDrill only extracts threat actions and maps them to TTP categories, while EXTRACTOR only extracts subject-predicate-object triplets that involve IOCs. THREATKG, on the other hand, can extract a wider range of knowledge types, such as various entities (IOCs, threat actors, malware, etc.), relations, and entity attributes, from various OSCTI sources. Another important difference between TTPDrill and EXTRACTOR and THREATKG is that TTPDrill and EXTRACTOR only extract limited knowledge from a single OSCTI report, while THREATKG can extract threat knowledge from a large number of OSCTI reports from various sources and construct a threat knowledge graph, through an automated system. This makes THREATKG much more comprehensive in capturing the threat landscape

As the types of entities covered by TTPDrill and EXTRACTOR are very limited, in our evaluations, we only compare the relations extracted by TTPDrill and EXTRACTOR with THREATKG. The results are shown in Table VI. We can observe that: (1) EXTRACTOR has a lower performance than THREATKG in relation extraction, because EXTRACTOR can only extract relations between IOCs. (2) TTPDrill suffers from a low precision, because it aims to extract threat actions and map them to TTP categories, so it extracts many phrases that may not be relevant to the relation types.

7.4 RQ4: Runtime Performance

We measured a single-process procedure for all OSCTI reports. The evaluation took 87.3 hours to nish, reaching a processing throughput of 24.7 reports per minute. With 11 articles added to the system every day, the expected daily workload is less than 30 seconds. We show a performance breakdown analysis in Table VII. We notice that the extractors take most of the time and the dependency parsing is the bottleneck. A potential reason is that the sentence-wise dependency parsing for long content OSCTI report is time-consuming. As evidence, the dependency parsing for the source apt_notes with an average content length of 32503 characters takes 22.0 seconds on average (88.5% of total processing time for that source). In contrast, for the source symantec_vulnerability with an average content length of 332 characters, it takes 0.1 seconds on average (71.5% of total processing time for that source). These results show that THREATKG is ef cient enough for real-world use cases.

8 Related Work

OSCTI services and platforms. Various platforms and services have been created to manage OSCTI. Platforms like AlienVault OTX [10], IBM X-Force [12], PhishTank [19], MISP [15], and OpenCTI [17] allow users to contribute, share, or manage OSCTI. Unlike these platforms that require user contribution, THREATKG gathers and aggregates threat knowledge automatically using AI-based techniques. There are also platforms, such as OpenPhish and Abuse.ch [9], that collect threat knowledge automatically. However, they only focus on speci c types of entities: ThreatMiner focuses on low-level IOCs (e.g., IPs, domains, and URIs). OpenPhish focuses on phishing URLs. Abuse.ch focuses on malware and botnets. In addition, several studies have been proposed to better analyze OSCTI reports, such as understanding vulnerability reproducibility [18] and measuring threat knowledge quality (e.g., consistency, accuracy, and coverage) [29, 37]. Such research is orthogonal to THREATKG.

OSCTI formats and ontologies. There exist open standard formats such as STIX [8] and OpenIOC [2] for exchanging threat intelligence. They are schemas rather than the large knowlede graph constructed by THREATKG that contains the actual knowledge. The

Table VI: Relation extraction performance (-#false negatives, +#false positives) of TTPDrill, EXTRACTOR, and THREATKG

CTI reports	# Words	# Manually Labeled Relations	TTPDrill	EXTRACTOR	THREATKG
hunting-for-linux-library-injection-with-osquery	1431	145	(-105, +341)	(-128, +207)	(-47, +47)
android-backdoor-disguised-as-a-kaspersky-mobile-security-app-65534	233	34	(-28, +130)	(-28, +43)	(-16, +16)
peer-peer-poisoning-attack-against-kelihosc-botnet	829	43	(-38, +214)	(-31, +116)	(-6, +6)
dragon y-energy-companies-sabotage	1095	104	(-88, +619)	(-84, +221)	(-22, +22)
android-apps-coronavirus-covid19-malicious	388	30	(-24, +105)	(-21, +59)	(-7, +7)
new-versions-of-the-iexplorer-zeroday-emerge-targeting-defence-and-industr	317	40	(-33, +99)	(-30, +55)	(-15, +15)
Trojan.Win64.Shelma	17	2	(-0, +13)	(-1, +1)	(-0, +0)
analyzing-the-security-of-ebpf-maps	1066	105	(-89, +533)	(-86, +217)	(-31, +31)
security-vpn-ios-macos	400	12	(-9, +187)	(-8, +112)	(-2, +2)
duqu-next-stuxnet	502	49	(-44, +266)	(-38, +89)	(-18, +18)
Trojan-DDoS.Win32.Nesmed	12	4	(-4, +6)	(-3, +1)	(-2, +2)
inside-geinimi-android-trojan-chapter-one-encrypted-data-and-communication	855	11	(-11, +31)	(-10, +56)	(-1, +1)
beapy-cryptojacking-worm-china	1271	123	(-104, +508)	(-102, +316)	(-34, +34)
a-few-words-about-the-hlux-botnet-29806	51	11	(-11, +41)	(-5, +11)	(-2, +2)
google-cloud-platform-security-monitoring-with-usm-anywhere	542	47	(-37, +289)	(-33, +129)	(-11, +11)
Alienvault_Scanbox	400	28	(-22, +180)	(-16, +85)	(-6, +6)
Overall Precision			0.038	0.198	0.85
Overall Recall			0.199	0.305	0.85
Overall F-1 Score			0.063	0.217	0.85

Table VII: Runtime performance breakdown

Stage	Total Processing Time (h)	Percentage		
Porter	0.54			0.6%
Checker	0.03			0.0%
Parser	1.45			1.7%
Extractor	85.26	97.7%	Content relevance analysis	2.1%
			Dependency parsing for IOC relation extraction	83.1%
			BiLSTM CRF entity extraction recognition	6.0%
			Potential relation marking	0.9%
			PCNN-ATT relation extraction	5.7%

knowledge gathered by THREATKG can be easily converted into these formats for distribution. MITRE ATT&CK [16] is a knowledge base for cyber adversary behaviors based on real-world observations. It is manually curated by security experts and does not focus on automated knowledge extraction from OSCTI reports as done in THREATKG. It also does not contain IOC relations. There are some cyber ontologies [31, 41, 45, 46] that support reasoning, but most of them only focus on sub-domains of threat knowledge, such as IDS [41, 46] and malware behavior [31]. None of these works focus on automated threat knowledge extraction from natural language text.

Security information extraction. Several works have been proposed to extract threat knowledge from text. iACE [38] extracts IOCs from security articles using a graph mining technique. ChainSmith [49] further classifies the extracted IOCs into different attack campaign stages (e.g., baiting, exploitation, installation, and C&C) using neural networks. TTPDrill [33] extracts threat actions from Symantec reports and maps them to pre-defined attack patterns. EXTRACTOR [44], ThreatRaptor [30], and HINTI [48] use various NLP techniques to extract IOC entities and IOC relations. These work focus extract only IOCs or IOC relations from a single OSCTI report. In contrast, THREATKG extracts a wider range of entities (e.g., threat actors, techniques, tools) and relations from multiple reports to construct a threat knowledge graph.

9 Conclusion

We presented THREATKG, a system for automated open-source cyber threat intelligence gathering and management. THREATKG automatically constructs a threat knowledge graph from OSCTI reports using AI-based techniques. In future work, we aim to explore other types of security applications that can be enabled by THREATKG, such as intrusion detection and cyber threat hunting.

Acknowledgement. This work was supported in part by the 2021 Cisco Research Award and the Commonwealth Cyber Initiative (CCI). Any opinions, findings, and conclusions made in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] [n. d.]. Word2Vec document. https://code.google.com/archive/p/word2vec/.
[2] 2013. The History of OpenIOC. https://www.fireeye.com/blog/threat-research/2013/09/history-openioc.html.
[3] 2014. Target Data Breach Incident. http://www.nytimes.com/2014/02/27/business/target-reports-on-fourth-quarter-earnings.html?_r=1.
[4] 2015. The CozyDuke APT. https://securelist.com/the-cozyduke-apt/69731/.
[5] 2017. Symantec Threat Intelligence. https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence.
[6] 2020. The Equifax Data Breach. https://www.ftc.gov/equifax-data-breach.
[7] 2021. Ransom.Win32.LOCKBIT.YEBGW. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/ransom.win32.lockbit.yebgw.
[8] 2021. Structured Threat Information eXpression. http://stixproject.github.io/.
[9] 2022. Abuse.ch. https://abuse.ch/.
[10] 2022. AlienVault OTX. https://otx.alienvault.com/.
[11] 2022. APTnotes. https://github.com/aptnotes/data.
[12] 2022. IBM X-Force Exchange. https://exchange.xforce.ibmcloud.com/.
[13] 2022. ioc-parser. https://github.com/armbues/ioc_parser.
[14] 2022. Kaspersky Threat Encyclopedia. https://threats.kaspersky.com/.
[15] 2022. MISP - Open Source Threat Intelligence Platform & Open Standards For Threat Information Sharing. https://www.misp-project.org/.
[16] 2022. MITRE ATT&CK. https://attack.mitre.org.
[17] 2022. OpenCTI. https://www.opencti.io/en/.
[18] 2022. OpenPhish. https://openphish.com/.
[19] 2022. PhishTank. https://www.phishtank.com/.
[20] 2022. Schneier on Security. https://www.schneier.com/.
[21] 2022. SecureList. https://securelist.com/.
[22] 2022. Snorkel. https://snorkel.org.
[23] 2022. Sophos News. https://news.sophos.com/en-us/.
[24] 2022. Trend Micro Threat Encyclopedia. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/.
[25] 2023. Webroot. https://www.webroot.com/blog/.
[26] 2024. Demo Video of Our GUI Application for Threat Knowledge Graph Exploration. https://youtu.be/wR4TdK7uc_U.

[27] 2024. Demo Video of Our QA System for Threat Knowledge Acquisition. https://youtu.be/6IDPQGMwgvM.

[28] Josh Barnes and Piet Hut. 1986. A hierarchical O(N log N) force-calculation algorithm. nature (1986).

[29] 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 28th USENIX Security Symposium (USENIX Security 19). 869–885.

[30] Peng Gao, Fei Shao, Xiaoyuan Liu, Xusheng Xiao, Zheng Qin, Fengyuan Xu, Prateek Mittal, Sanjeev R Kulkarni, and Dawn Song. 2021. Enabling ef cient cyber threat hunting with cyber threat intelligence. In 2021 IEEE 37th International Conference on Data Engineering (ICDE). 193–204.

[31] André Grégio, Rodrigo Bonacin, Olga Nabuco, Vitor Monte Afonso, Paulo Lício De Geus, and Mario Jino. 2014. Ontology for Malware Behavior: A Core Model Proposal. In 2014 IEEE 23rd International WETICE Conference (WETICE). 453–458.

[32] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation 9, 8 (1997), 1735–1780.

[33] 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 Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC). 103–115.

[34] Dan Jurafsky. 2000. Speech & language processing. Pearson Education India.

[35] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL) 260–270.

[36] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end Neural Coreference Resolution. In The 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP). 188–197.

[37] 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 28th USENIX Security Symposium (USENIX Security 19). 851–867.

[38] 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 Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS). 755–766.

[39] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019).

[40] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL). 1003–1011.

[41] Sumit More, Mary Matthews, Anupam Joshi, and Tim Finin. 2012. A Knowledge-Based Approach to Intrusion Detection Modeling. In 2012 IEEE Symposium on Security and Privacy Workshops (S&P Workshops). 75–81.

[42] 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 27th USENIX Security Symposium (USENIX Security)

[43] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. Advances in neural information processing systems (NeurIPS) 29 (2016), 3567–3575.

[44] Kiavash Satvat, Rigel Gjomemo, and VN Venkatakrishnan. 2021. EXTRACTOR: Extracting attack behavior from threat reports. In 2021 IEEE European Symposium on Security and Privacy (EuroS&P). 598–615.

[45] Zareen Syed, Ankur Padia, Tim Finin, Lisa Mathews, and Anupam Joshi. 2016. UCO: A uni ed cybersecurity ontology. UMBC Student Collection (2016).

[46] J. Undercofer, Anupam Joshi, Tim Finin, and John Pinkston. 2003. A Target-Centric Ontology for Intrusion Detection. In Workshop on Ontologies in Distributed Systems (2003).

[47] Daojian Zeng, Kang Liu, Yubo Chen, and Jun Zhao. 2015. Distant supervision for relation extraction via piecewise convolutional neural networks. In Proceedings of the 2015 conference on empirical methods in natural language processing (EMNLP) (2015), 1753–1762.

[48] Jun Zhao, Qiben Yan, Xudong Liu, Bo Li, and Guangsheng Zuo. 2020. Cyber Threat Intelligence Modeling Based on Heterogeneous Graph Convolutional Network. In 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID). 241–256.

[49] Ziyun Zhu and Tudor Dumitras. 2018. ChainSmith: Automatically Learning the Semantics of Malicious Campaigns by Mining Threat Intelligence Reports. In 2018 IEEE European Symposium on Security and Privacy (EuroS&P). 458–472.

Table VIII: List of OSCTI sources

OSCTI Source	Number of Reports	Type	URL
apt_notes	539	APT Reports	https://github.com/aptnotes/data
attcybersecurity	244	Enterprise Security Blog	https://cybersecurity.att.com
ciscoumbrella	478	Enterprise Security Blog	https://umbrella.cisco.com
cloudflare	1,791	Enterprise Security Blog	https://blog.cloudflare.com
crowdstrike	942	Enterprise Security Blog	https://www.crowdstrike.com
csoonline	1,258	Enterprise Security Blog	https://www.csoonline.com/
darknet	2,107	Enterprise Security Blog	https://www.darknet.org.uk
fireeye	209	Enterprise Security Blog	https://www.fireeye.com
forcepoint	1,190	Enterprise Security Blog	https://www.forcepoint.com
hotforsecurity	9,496	Enterprise Security Blog	https://hotforsecurity.bitdefender.com
kasperskydaily	3,350	Enterprise Security Blog	https://www.kaspersky.com
krebsonsecurity	2,129	Personal Security Blog	https://krebsonsecurity.com
malwarebytes	3,382	Enterprise Security Blog	https://blog.malwarebytes.com
mcafee	6,295	Enterprise Security Blog	https://www.mcafee.com
nakedsecurity	14,653	Enterprise Security Blog	https://nakedsecurity.sophos.com
nccgroup	520	Enterprise Security Blog	https://research.nccgroup.com
paloalto	3,284	Enterprise Security Blog	https://blog.paloaltonetworks.com
recordedfuture	1,537	Enterprise Security Blog	https://www.recordedfuture.com
rsa	71	Enterprise Security Blog	https://www.rsa.com
securelist	5,630	Enterprise Security Blog	https://securelist.com
shneieronsecurity	8,110	Personal Security Blog	https://www.schneier.com
sophos	1,822	Enterprise Security Blog	https://news.sophos.com
spiderlabs	1,401	Enterprise Security Blog	https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/
symantecthreatintelligence	177	Enterprise Security Blog	https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/
thehackernews	8,432	Enterprise Security Blog	https://thehackernews.com
threatpost	5,427	Enterprise Security Blog	https://threatpost.com/
trendmicro	2,393	Enterprise Security Blog	https://blog.trendmicro.com
trendmicrosecurityintelligence	4,001	Enterprise Security Blog	https://blog.trendmicro.com/trendlabs-security-intelligence
trustwave	571	Enterprise Security Blog	https://www.trustwave.com/en-us/resources/blogs/trustwave-blog/
unit42_paloalto	645	Enterprise Security Blog	https://unit42.paloaltonetworks.com/
webroot	1,438	Enterprise Security Blog	https://www.webroot.com
welivesecurity	5,780	Enterprise Security Blog	https://www.welivesecurity.com
zscaler	770	Enterprise Security Blog	https://www.zscaler.com
malwarebytes	163	Threat Encyclopedia	https://blog.malwarebytes.com
symantec_threats	37,588	Threat Encyclopedia	http://asb-sngweb.symantec.com
symantec_vulnerabilities	7,431	Threat Encyclopedia	http://asb-sngweb.symantec.com
kaspersky_threat	1,430	Threat Encyclopedia	https://threats.kaspersky.com/en
kaspersky_vulnerability	1,968	Threat Encyclopedia	https://threats.kaspersky.com/en
trendmicro_malware	534	Threat Encyclopedia	https://www.trendmicro.com
trendmicro_spam	396	Threat Encyclopedia	https://www.trendmicro.com
fsecure	4,083	Threat Encyclopedia	https://www.f-secure.com

Table IX: List of relation classes that THREATKG covers

No	Relation Class	Explanation	Indicator Verbs
1	use	A bad actor, malware, or technique uses something to finish a goal. The action is general, without much detail.	use, take, utilize, employ
2	execute	An actor executes a specific tool, program, function, etc.	perform, parse, execute, conduct, run, calculate, carry out, call, initiate, launch
3	enable	A tool/technique enables one thing means this tool/technique makes this thing possible.	tunnel, allow, rely, provide, sign, attribute, harden,activate
4	own	One thing owns something means such thing contains something or is composed by something.	compose, include, consist, contain, inside
5	inject	Typically "foo injects bar" means a bad actor foo inserts something malicious, bar into the targets.	save, load, attack, install, write, embed, upload, inject, deploy, infect
6	alter	"foo alters bar" means foo modifies or changes something bar on the targets to achieve some malicious goals.	change, define, affect, compromise, change, configure, tamper, redirect
7	get	The subject obtains some information, data, etc.	decrypt, retrieve, extract, download, obtain, send, receive, steal, access
8	keep	To make something consistent by remaining or storing something.	persist, maintain, remain, store, host
9	spread	To duplicate and send one thing (typically malware) from one place to other places.	spread, circulate, distribute, release, share, duplicate, propagate
10	hide	Foo hides bar means foo makes bar unseeable or undetectable.	encrypt, hide, obfuscate
11	relate	One thing has some relations or communications with another thing.	link, match, relate, associate, communicate, connect, alias
12	create	To generate or make something that did not exist before.	compute, craft, create,build
13	update	Neutral word, typically means to update a program to a new version, or update the state.	modify, recreate, restructure
14	break	Foo breaks bar means foo stops or prevents bar	delete, block, destroy, stop, circumvent, bypass, drop
15	find	To discover or locate the desired things from the whole dataset.	select, find, search, detect, look for, scan
16	mitigate	Typically means to alleviate bad influence.	mitigate, resolve, protect
17	aim	A bad actor/malware aims at one thing means this thing is the target or victim.	aim, target, attack, for