2024 IEEE 10th International Conference on Collaboration and Internet Computing (CIC) | 979-8-3503-8670-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/CIC62241.2024.00012

GRAPHENE: Towards Data-driven Holistic Security Posture Analysis using AI-generated Attack Graphs

Charalampos Katsis
Purdue University
ckatsis@purdue.edu

Xin Jin
The Ohio State University
jin.967@osu.edu

Fan Sang
Georgia Institute of Technology
fsang@gatech.edu

Jiahao Sun
Georgia Institute of Technology
jiahaosun@gatech.edu

Elisa Bertino
Purdue University
bertino@purdue.edu

Ramana Rao Kompella
Cisco Research
rkompell@cisco.com

Ashish Kundu
Cisco Research
ashkundu@cisco.com

***Abstract*—The rampant occurrence of cybersecurity breaches imposes substantial limitations on the progress of network infrastructures, leading to compromised data, financial losses, potential harm to individuals, and disruptions in essential services. The current security landscape demands the urgent development of a holistic security assessment solution that encompasses vulnerability analysis and investigates the potential exploitation of these vulnerabilities as attack paths. In this paper, we propose GRAPHENE, an advanced system designed to provide a detailed analysis of the security posture of computing infrastructures. Using user-provided information, such as device details and software versions, GRAPHENE performs a comprehensive security assessment. This assessment includes identifying associated vulnerabilities and constructing potential attack graphs that adversaries can exploit. Furthermore, it evaluates the exploitability of these attack paths and quantifies the overall security posture through a scoring mechanism. The system takes a holistic approach by analyzing security layers encompassing hardware, system, network, and cryptography. Furthermore, GRAPHENE delves into the interconnections between these layers, exploring how vulnerabilities in one layer can be leveraged to exploit vulnerabilities in others. In this paper, we present the end-to-end pipeline implemented in GRAPHENE, showcasing the systematic approach adopted for conducting this thorough security analysis.**

***Index Terms*—Attack Graphs, CVE, CWE, Attack Paths, Risk Analysis, Security Posture Analysis**

I. INTRODUCTION

The escalating complexity of enterprise networks and the proliferation of applications and software packages from diverse sources have resulted in environments that are highly susceptible to cyber-attacks. The intricate network of interconnected devices, coupled with technologies such as cloud computing and Internet of Things (IoT) devices, further expands the attack surface, providing attackers with numerous entry points [1]. Moreover, the interconnected nature of networks intensifies the impact of a single vulnerability, enabling adversaries to navigate interconnected systems and compromise multiple hosts and devices. Finally, hundreds of vulnerabilities are disclosed monthly in the national vulnerability databases; thus, an approach for evaluating and understanding their impact is critical for several reasons, such as prioritizing patching efforts.

Problem Scope. Given the expanding attack surface, we need comprehensive systems able not only to identify vulnerabilities specific to the infrastructure of interest but also to discern how these vulnerabilities can be exploited in sequences. Our goal is to develop a solution that leverages *attack graphs* to (1) understand how vulnerabilities might serve as a sequence of steps in a multi-step attack and (2) scrutinize and grasp the implications of each vulnerability on the infrastructure under analysis. Such a deep understanding of the nature and impact of each vulnerability is essential to tailor effective approaches for mitigation by subject matter experts. The system implementing the solution should also be able to identify and curate information regarding known vulnerabilities across applications, systems and devices.

Challenges. Designing such a system requires addressing the following challenges: **(C1)** The vulnerabilities are typically described in natural language (i.e., common vulnerabilities and exposures – CVE) rather than in a formally defined encoded format. Therefore, a systematic approach is needed to capture the vulnerability semantics and convert them into a suitable format for further analysis. **(C2)** The construction of attack paths constituting a chain of vulnerabilities to the attacker's objectives requires extracting the conditions that allow one to exploit a vulnerability (i.e., *preconditions*) and the state of the system once a vulnerability is exploited (i.e., *postconditions*). **(C3)** It is hard to develop security quantification metrics that capture the criticality of vulnerabilities and the impact on the system under analysis.

Limitations of Prior Works. Previous research on attack graph generation has several limitations. Many approaches require manually inputting infrastructure-related CVEs [2]–[4], which is challenging due to the constant emergence of new vulnerabilities. A single vulnerability can substantially alter the security landscape by introducing new attack possibilities or high-risk pathways. Additionally, some methods rely on proprietary formal expressions for defining vulnerabilities [2],

The first three authors contributed equally to this work. Katsis, Jin, Sang and Sun did part of the work during an internship at Cisco Research.

979-8-3503-8670-7/24/$31.00 ©2024 IEEE
DOI 10.1109/CIC62241.2024.00012

[3], [5], which must be frequently updated to reflect new attack vectors, and the accuracy of these definitions directly impacts the quality of the attack graph. Some approaches use hard-coded heuristics and keyword matching to connect CVEs [2], [4]–[6], which limits applicability across different vulnerabilities and fails to capture the nuanced information in natural language text. Finally, many risk analysis methods do not assess risk with respect to the underlying infrastructure [7]–[10]. Therefore, a tailored vulnerability assessment is crucial for fully addressing the specific risks posed to an infrastructure.

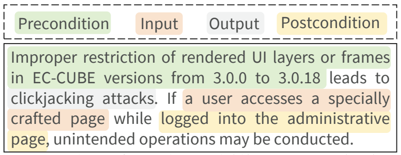
Our Approach. To address such challenges, we design an innovative solution called GRAPHENE, which serves as a comprehensive security posture analyzer for computing infrastructures. Users (e.g., security officers) provide an inventory of the interconnected devices' details and software versions installed. Then, the system continuously monitors trustworthy data sources, such as the NIST National Vulnerability Database (NVD) [11], for vulnerabilities specific to components of a given infrastructure.

Using named entity recognition (NER), GRAPHENE is able to extract the semantic meaning of these vulnerabilities and encode them into a latent space for in-depth analysis. In particular, it automatically extracts the preconditions required for an adversary to exploit a vulnerability and the result after exploiting the vulnerability (i.e., postconditions). For example, GRAPHENE can discern that exploiting a particular CVE might require the presence of the TensorFlow XLA compiler in the Google TensorFlow version prior to 1.7.0 as a precondition. Simultaneously, it captures postconditions such as a system crash resulting from a heap buffer overflow.

GRAPHENE uses a semantic similarity-matching approach based on word embeddings [12] to link the postconditions of one vulnerability with the preconditions of another, constructing potential attack graphs for a given topology. After generating these graphs, it evaluates the security posture using a novel layered analysis. Vulnerabilities are grouped into distinct layers—such as machine learning, system, hardware, network, and cryptography—based on keyword matching and CWE information. GRAPHENE performs similarity matching within and across these layers, uncovering how vulnerabilities in one layer can exploit those in another. This layered approach enables subject matter experts to prioritize risk analysis, develop targeted mitigation strategies, and implement patches based on the severity of vulnerabilities in each layer.

As a result, GRAPHENE automatically produces two types of attack graphs: *cumulative* (or multi-layer) attack graphs and *layered* attack graphs. Cumulative attack graphs show how an attacker could exploit CVEs across multiple layers. In contrast, layered attack graphs focus on exploiting vulnerabilities within the same layer. This dual representation provides a comprehensive understanding of the potential attack paths across different layers and within individual layers.

GRAPHENE traverses the generated attack graphs to thoroughly analyze the infrastructure's security posture, offering both cumulative and layer-specific analytics. These analytics assess (1) the effort required by an adversary to exploit a step or sequence of steps and (2) the impact of those steps based on the criticality of the affected resources. After computing scores relative to this criticality, GRAPHENE highlights vulnerabilities and high-impact attack paths that demand immediate attention. The ultimate goal is to equip security officers with a clear understanding of their network's threat landscape, enabling organizations to proactively tailor mitigation strategies and strengthen defenses against evolving threats.


Precondition Input Output Postcondition

Improper restriction of rendered UI layers or frames in EC-CUBE versions from 3.0.0 to 3.0.18 leads to clickjacking attacks. If a user accesses a specially crafted page while logged into the administrative page, unintended operations may be conducted.

Fig. 1: An example of Vulnerability Description and Attack Graph Node Attributes for CVE-2020-5679.

Contributions. We make the following contributions:

- We introduce a novel fully-automated security posture analyzer to generate attack graphs for computing infrastructures.
- We propose a natural language processing approach based on NER and word embeddings that streamlines pre- and postcondition extraction, facilitating the generation of attack graphs without the need for manual intervention.
- Our framework adopts a comprehensive strategy for analyzing security postures in a multi-layered fashion, which analyzes each layer separately and combines them into one unified analysis.
- We propose risk scoring methods for tailored analysis of the underlying network infrastructure.

II. BACKGROUND

CVE and CWE. Common Vulnerabilities and Exposures (CVE) disclosures are essential in cybersecurity, providing a standardized system for identifying and cataloging known vulnerabilities with unique identifiers enabling prompt detection and response to threats. Complementing this, the Common Weakness Enumeration (CWE) system categorizes the weaknesses that cause vulnerabilities, offering a structured approach to understanding security issues. Additionally, the Common Vulnerability Scoring System (CVSS) assigns numerical scores (0 to 10) to CVEs, quantifying their severity based on impacts to confidentiality, integrity, and availability.

Attack Graphs. At the high level, an attack graph can be defined as a structured representation of the potential paths an attacker can take to compromise a network or system by exploiting vulnerabilities [2], [8]. In such a representation, the *attack graph nodes* are the attack units that assemble basic vulnerability attributes combined as coherent entities. These attributes encompass preconditions, postconditions, inputs, and outputs. *Preconditions* refer to a collection of system properties that must hold for an exploit to succeed. For example, in Figure 1, the CVE requires any EC-CUBE version from 3.0.0 to 3.0.18 to be installed. If these preconditions are not met, it becomes possible to render all subsequent steps of an attack ineffective. *Postconditions* are the system properties that hold as the results of an attack step, which are necessary for the generation of outputs. For example, in Figure 1, the user must be logged into the admin page for the clickjacking attack (output) to work. The *inputs* are the actions that attackers need to take to trigger the vulnerability and perform the exploit. In Figure 1, the user must access a specifically crafted page that serves as an input to the vulnerability. The *outputs* refer to the final values or results that the system returns or produces when exploits to vulnerabilities are executed. In Figure 1, the output of the CVE exploitation is the clickjacking attack.

Attack graph nodes can be categorized as attackers, attack targets, and vulnerabilities. Generally, attackers act as the source or root nodes, vulnerabilities function as intermediate nodes, and attack targets represent the sink/leaf nodes. By exploiting vulnerabilities, attackers can perform a sequence of steps or actions to attack the victims, such as gaining unauthorized access to a network or system. Such steps and actions are the edges in the attack graph, serving as the basic connections. Formally, *attack graph edges* represent the transitions and chains of vulnerabilities [13]. Successful attacks often require executing a series of exploits in a specific sequence. For example, to compromise the macOS Kernel through Safari, exploiting a chain of six vulnerabilities is required [14]. Therefore, the edge connecting two nodes in an attack graph indicates that the vulnerability exploited by one node can serve as the input and trigger for the other node.

III. Automated Attack Graph Generation

In this section, we present how we leverage machine learning and natural language processing to build attack graphs.

A. Attack Graph Node Identification

The attack graph nodes are the fundamental element of attack graphs, where they provide relevant vulnerability details, i.e., preconditions, postconditions, inputs, and outputs. Identifying attack graph nodes from vulnerability descriptions and reports is not trivial, as it requires understanding the semantics of vulnerabilities. Consider the description of CVE-2020-5679 shown in Figure 1, which can be exploited to perform clickjacking attacks. With manual efforts, human analysts can identify the key properties of this vulnerability by understanding the description and identifying the words or phrases that represent the nodes. However, manual identification of graph nodes based on human knowledge is not scalable. Another approach is to use pivot words [6] or heuristic rules [2], [4], [13], [15], [16]. However, these methods are not generalizable since natural language is noisy [17], e.g., different vulnerability descriptions can include semantically but syntactically different expressions.

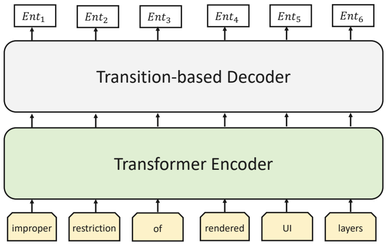


Fig. 2: The Named Entity Recognition Model

Node Identification by Entity Recognition. To address the aforementioned challenges, we propose to use NER for automatic attack graph node identification, which is a process of identifying entities in input texts by classifying words and phrases in vulnerability texts into the corresponding entities (addressing **C1**). Unlike existing approaches that have very limited scope and use weak models [18]–[20], e.g., only focusing on home computers [19], we propose a generalizable solution using generative language models. Specifically, as shown in Figure 2, our model takes as input the vulnerability descriptions and documents, which are tokenized into natural language tokens. These tokens are then encoded by a transformer encoder. The encoder generates semantic embeddings by mapping the input tokens into latent space. The transition-based decoder maps the token embeddings into named entities based on a finite-state transducer [21]. The model is expected to identify security entities. In this paper, we define six entities, including vulnerability type, affected product, root cause, impact, attacker type, and attack vector. These entities are selected because they are the required information by the NVD CVE maintenance team and rigorously defined in the official CVE template for *all CVE descriptions* that covers the critical details for automated phrasing [22]. The design choice of selecting these entities as model output makes our model general to vulnerabilities because of the wide adoption of the MITRE CVE template. We systematically categorize the identified entities as attributes of attack nodes, namely preconditions, postconditions, inputs, and outputs. In this framework, we classify the affected product entity as a precondition, the vulnerability type as a postcondition, and the attacker type and root cause as inputs. Finally, the impact and attack vector entities are categorized as outputs.

B. Attack Graph Edge Connection

The attack graph edge indicates that the exploit result of one vulnerability can serve as the trigger for another vulnerability. Building the edge connection requires the precise reasoning of relationships between two nodes. Given the identified conditions in natural language, a simple way to build edges is to match the words of different graph nodes. Yet, it may not always work since morphological words (e.g., synonyms, abbreviations, and misspellings) are widely used in vulnerability descriptions and texts. For instance, in the case of web injection vulnerabilities, the abbreviation "XSS" is frequently employed to refer to cross-site scripting. To overcome such limitations, we propose to utilize word embeddings to semantically build attack graph edges.

Word Embedding-based Edge Construction. In natural languages, words in different contexts can carry different meanings. For example, the word "band" has different meanings under material and music context and corpus. In other words, the word embeddings trained on one specific domain

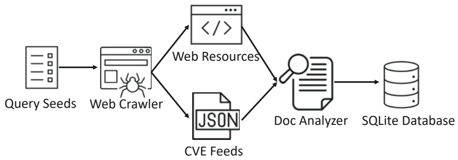


Fig. 3: The Security Corpus Curation Framework.

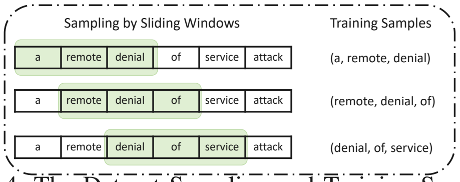


Fig. 4: The Dataset Sampling and Training Samples.

cannot be directly used in other domains because there can be semantic changes. Therefore, we argue that existing word vectors (e.g., Glove and word2vec [23], [24]), which are pre-trained on general English corpus (e.g., Wikipedia), cannot precisely deliver word semantics in the security domain.

In order to obtain embeddings tailored for holistic security assessment, we propose to train embedding models using a corpus specifically focused on security (addressing **C2**). This approach offers two distinct advantages: (1) The resulting word embeddings will facilitate the accurate semantic matching of attack nodes, allowing for precise identification and classification. (2) By quantifying the matching outcomes as similarity scores, we can assign weights to the attack edges, enabling a more nuanced representation of each attack's severity or relevance. To obtain the similarity scores, we first compute the semantic representations of attack node attributes by averaging the vectors of all words in the ports. Next, we calculate the node similarity by the cosine similarity function.

To train the word embeddings, we curate the representative security corpus from vetted sources, e.g., NVD [11]. The NVD database contains detailed vulnerability descriptions and references of CVEs that have undergone thorough manual assessment and processing. Moreover, there are also many other online resources, such as MITRE ATT&CK knowledge base [25], that collect comprehensive vulnerability information, such as CWEs, which we also consider in our dataset curation process. We build a security corpus curation framework as shown in Figure 3. The framework takes a list of query seeds (e.g., CVEs and keywords) as input. The output of the web crawler includes online web pages and resources and the CVE feeds. Note that the CVE feeds contain much useful information, such as CWEs, vulnerable products, affected versions, CVSS scores, and reference links [26]. After obtaining all the online resources and CVE feeds, we run the doc parser to clean up these documents and extract useful content, such as the CVE descriptions and vulnerability texts.

To preprocess our curated dataset, we first clean it by removing punctuation. We do not follow the common natural language processing practice of removing stop words, as some of these words play a key role in the security context. For example, removing the stop word "of" from the phrase "denial of service" will break this popular attack phrase and change its semantics. Next, we sample the processed security documents into training samples using sliding windows as shown in Figure 4. The training samples are sets of center words and context words. For example, in the word set `(a, remote, denial)`, `remote` is the center word while `a` and `denial` are the context words. This sampling method enables us to learn the word semantics by modeling the context information. For example, the word embedding model Continuous Bag-of-Words (CBOW) [27] learns word semantics by optimizing the probability likelihood estimation:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{-c\leq j\leq c, j\neq 0} log(p(w_i \mid w_{i+j})) \tag{1}$$

where N is the total number of words in the security document, c is the sliding window size, and w_i and w_{i+j} are the center word and context word.

C. Attack Graph Construction and Partition

The attack graph comprises three primary node types: the attacker (source nodes), existing CVEs associated with different network entities (intermediate nodes), and CWEs serving as attacking targets (sink nodes). The construction of edges connecting nodes and assigning corresponding edge weights depend on the types of connected nodes and available data. By default, an edge is established from the attacker node to each CVE node, assuming the attacker can exploit the CVE. This inclusivity aligns with GRAPHENE's objective to generate all conceivable scenarios under various adversarial assumptions in the network. For example, if a CVE's attack vector necessitates physical access to the device, GRAPHENE includes an analysis of such a scenario. Users can subsequently filter the generated attack graphs based on their network's specific adversarial assumptions. The edge weights for these edges are predominantly determined by CVSS base scores, indicating the likelihood of a successful attacker exploit, directly correlating with the severity of the threat posed by exposure to such vulnerabilities.

For any given pair of CVE nodes, an edge is established if the postcondition of one CVE node aligns (partially) with the precondition of another CVE node, indicating the potential for an attacker to exploit one vulnerability to access the other—essentially forming a chain of vulnerabilities for exploitation. Beyond relying on CVSS scores, the weights assigned to these edges are contingent on the node matching score of the two nodes, derived from the word-embedding-based node matching. The node matching score gauges the semantic similarity between the postcondition and precondition of the directed edge connecting two CVE nodes, offering insights into the difficulty an attacker might face in reaching the latter CVE by exploiting the former. Additionally, to mitigate graph complexity arising from marginally related CVE nodes, the user can stipulate the construction of edges only when the node matching score exceeds a specified threshold. Consequently, the resulting attack graph comprises paths that are more feasible for attackers, optimizing computational resources by excluding less viable paths. Lastly, each CVE node is linked to CWE nodes, representing the system's encountered threats. In the CVSS database, each CVE is associated with one or several corresponding CWEs, each instantiated as an edge in the attack graph. Similar to other edge types, the CVSS scores determine the edge weights, with the option to prune edges by setting a weight threshold.

Attack Graph Partition. After obtaining the holistic security posture, another potential need is to get an in-depth security analysis by focusing on the specific layer or scope of the target systems. To achieve this, we propose the attack graph partition component to get subgraphs from the cumulative attack graph.

GRAPHENE follows two approaches to classify the identified vulnerabilities to the layers of interest. In the first approach, it uses a predefined set of high-frequency keywords that typically appear in the corresponding layer. For example, in the network layer, the system uses keywords such as "TCP", "SSL", and "certificate". The second approach is based on the MITRE's CWE identifier appointed by the vulnerability database. The reason for using both approaches is that, on the one hand, only keyword matching may result in misclassified vulnerabilities. On the other hand, relying merely on the CWEs is not enough, as many vulnerabilities have not been appointed to CWEs. Additionally, the Dashboard service (Section V) allows an administrator to change the layer to which a particular vulnerability has been assigned.

IV. RISK SCORING SYSTEM

Following the construction of attack graphs, a comprehensive assessment of the security posture is conducted through a risk-scoring system (addressing **C3**). This system receives the attack graphs generated by Graph MS and produces analytics on the security posture.

Within the graphs, each CVE node is linked to exploitability, impact, and risk scores. These scores are derived according to the CVSS standard [28] (v. 3.1). Recognizing that these scores offer a limited, isolated perspective on vulnerability impact, GRAPHENE enhances the computed scores based on CVSS standards and integrates them for the risk assessment of attack graphs, considering the criticality of affected resources. The evaluation of security posture involves:

Computing graph exploitability, risk, and impact scores. The first step is to compute the scores based on the attack graphs.

Edge Exploitability Score (EES). Let ees_i denote the EES of the edge i. It is computed as follows:

$$ees_i = eScore(source(i)) + c \dot{\sum}_{x \in in\_edges(source(i))} ees_x \tag{2}$$

$source(i)$ returns the source node of edge i while $eScore()$ returns the exploitability score as a function of the node provided as an argument. $in\_edges()$ returns all inbound edges to the node provided as an argument, and c is a predefined multiplication constant ($c = 0.1$ in the experiments). Hence, the calculated EES for each edge is associated with the exploitability score of the edge's source node and the scores of all preceding edges in all the paths that include the edge.

Edge Impact Score (EIS). Let eis_i denote the EIS of edge i. It is computed as follows:

$$eis_i = iScore(sink(i)) + k \cdot \sum_{x \in out\_edges(sink(i))} eis_x \tag{3}$$

$sink(i)$ returns the sink node of edge i while $iScore()$ returns the impact score as a function of the node provided as an argument. $out\_edges()$ returns all the outbound edges from the node provided as an argument, and k ($k = 0.01$ in the experiments) is a predefined multiplication constant. Hence, the computed EIS of each edge is associated with the impact score of the edge's sink node and the scores of all the subsequent edges in all the paths that include the edge. In simpler terms, the EIS score of an edge accumulates the impact of all subgraphs starting with that edge.

The functions $eScore()$ and $iScore()$ in GRAPHENE are intentionally user-defined to account for the varying impact of CVEs across different deployment scenarios. For example, a CVE affecting a device in critical infrastructure has a much different impact than the same CVE in a publicly-facing demilitarized zone. To address this variability, users can create custom functions that factor in CVE scores and other relevant parameters, providing a tailored evaluation of exploitability and impact. If no custom functions are provided, GRAPHENE defaults to using the CVSS exploitability and impact scores for each CVE.

Edge Risk Score (ERS). Let ers_i denote the ERS of edge i. It is computed as $ers_i = weight_i \cdot (ees_i + eis_i)$, where $weight_i$ is the weight of edge i, and the values ees_i and eis_i are computed as per the Equations 2 and 3, respectively. Hence, the computed ERS of each edge is associated with the impact and exploitability scores of the edge i.

Once the computation of the edge scores is completed, we normalize them on a scale of 0 (low) to 10 (high). We apply the normalization for each set of edge scores (i.e., EES, EIS, ERS) as $\frac{10 \cdot edge\_score}{max\_set\_score}$, where $edge\_score$ is the original edge score, and max_set_score is the maximum score in the set.

Graph Scores. For each generated graph, GRAPHENE computes the exploit, impact, and risk scores. Each score is computed as the average of the EES, EIS, and ERS sets.

Identifying the shortest paths with respect to the attacker goals. We define the shortest path towards the attacker's goals as the path in which the sum of the edge scores present in the path is the highest. Given a graph, we first find the maximum exploitability score of all the nodes in the graph. We, then, define the weight of the edge i as $edge\_weight_i = max\_exploit - eScore(source(i))$, where $max\_exploit$ is the maximum exploitability score of all the nodes in the graph. Thus, the higher the exploitability score of the source node of an edge i, the lower the weight assigned to the edge. We then add a temporary node to the graph where all the sink nodes (i.e., the attacker goals) have an edge targeting it. This node is called the "supersink" node. Finally, we run a weighted shortest path algorithm from the attacker node to the sink

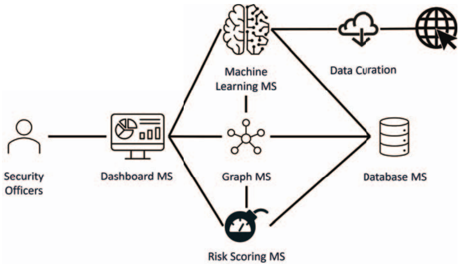


Fig. 5: GRAPHENE's microservices (MS) decomposition.

node. The algorithm runs in polynomial time and finds the paths with the least weight, thus the highly exploitable paths.

Identifying the high severity attack paths. To compute all the paths from the attacker node to the attacker's goals, GRAPHENE searches for all the possible paths up to a certain number of edges; this is the cutoff limit for path explorations. For each computed path, it computes the exploitability, impact, and risk score of every path, which is essentially the total sum of EES, EIS, and ERS of the edges in the path. Then GRAPHENE sorts the paths in descending order of risk, exploitability, and impact. GRAPHENE allows the system administrator to change how sorting is performed through the dashboard interface.

Identifying the key vulnerabilities that require immediate patching. To find those vulnerabilities in the attack graph, we measure the degree of every intermediate node (i.e., any node other than the source or sinks) in the graph. The degree of the node is defined as the number of edges that are incident to the node. A node with a high degree essentially means that the node is present in several attack paths. Thus, eliminating (i.e., patching) such nodes may render several attacks impractical.

Identifying the minimum set of vulnerabilities that cover all the attack paths. We apply a minimum set of vertex cover on the constructed attack graph to identify the minimum set of vulnerabilities that cover all the edges in the graph. As vertex cover is an NP-hard problem, we use a local-ratio approximation algorithm to find the minimum set vertex cover [29]. Thus, GRAPHENE identifies the minimum set of vulnerabilities that could render the attack paths impractical once mitigated.

V. IMPLEMENTATION

GRAPHENE is comprised of five microservices (MS) (see Figure 5). Steps 1 and 2 are implemented by the Machine Learning MS, step 3 by Graph MS and step 4 by the Risk Scoring MS. Those services are implemented with 987 lines of Python code. We build the named entity recognition and word embedding models based on spaCy [30], Gensim [31], and Scikit-learn [32]. For the risk scoring system, we use the NetworkX [33] library for graph construction, graph traversal, and risk score computation.

TABLE I: Overall Results of Our NER Model and Baselines

Performance	Baselines		Our Model
	en_core_web_sm	en_core_web_lg	
Precision	95.88	95.2	**98.75**
Recall	96.07	96.65	**98.55**
F1 Score	95.97	95.92	**98.65**

The Dashboard MS orchestrates the GRAPHENE pipeline. It has been implemented using Flask [34] and Dash [35] libraries. The dashboard provides an interface allowing the user (e.g., security officers and network admins) to import details about the network infrastructure, such as the communicating entities and device details. Once the posture analysis is completed, the dashboard presents the generated graphs and the results of the risk analysis. For instance, Figure 6 shows a multi-layer attack graph generated, where the initial CVE node is categorized within the ML layer, while the subsequent CVE node is categorized within the domain of hardware and systems. The Database MS stores all the constructed graphs and their analysis in a Neo4j graph database [36].

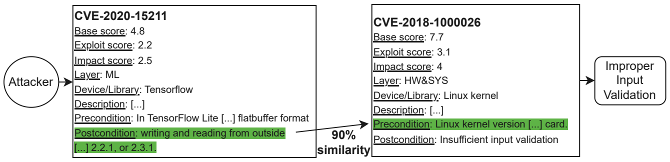

Attacker
CVE-2020-15211
Base score: 4.8
Exploit score: 2.2
Impact score: 2.5
Layer: ML
Device/Library: Tensorflow
Description: [...]
Precondition: In TensorFlow Lite [...] flatbuffer format
Postcondition: writing and reading from outside [...] 2.2.1, or 2.3.1.
90% similarity
CVE-2018-1000026
Base score: 7.7
Exploit score: 3.1
Impact score: 4
Layer: HW&SYS
Device/Library: Linux kernel
Description: [...]
Precondition: Linux kernel version [...] card.
Postcondition: Insufficient input validation
Improper Input Validation

Fig. 6: An example of a generated multi-layer attack graph.

VI. EVALUATION

In this section, we first evaluate the effectiveness of our NER-based model for processing CVE disclosures, followed by an evaluation of our word embedding technique used for attack graph construction. We then present a case study reasoning about the output of GRAPHENE. Our evaluations aim to answer the following research questions:

RQ1: How effective is the named entity recognition model for attack graph nodes?

RQ2: Can the proposed word embedding method semantically match the attack graph nodes?

We deploy GRAPHENE on a Google Cloud virtual machine with an Intel Broadwell CPU, 4 GB memory, 110 GB storage, the Ubuntu 20.04 operating system, and an NVIDIA Tesla P100 graphics card.

A. RQ1: Effectiveness of Named Entity Recognition Model

To answer **RQ1**, we evaluate our named entity recognition (NER) model by comparing it with baseline models on a large vulnerability NER dataset.

Dataset. Our dataset is adapted from PMA [37], which contains descriptions of 52,532 CVEs, 245,573 labeled entities, and 1,828,597 words. The entities have been annotated and classified into six categories, which are vulnerability type, affected products, root cause, impact, attacker type, and attack vector. It is worth noting that these entities are in line with the vulnerability definitions of the official CVE template [22]. We split the dataset into training, validation, and test sets based on the 8:1:1 split ratio.

Baselines and Evaluation Metrics. Although our NER model encoder is constructed on a transformer encoder, we showcase its superior adaptability by building baseline NER models using alternative encoders. Specifically, as our NER model

TABLE II: Performance on Individual Entities

Entity	Precision	Recall	F1 Score
Attacker Type	97.55	98.08	97.81
Impact	99.59	99.59	99.59
Attack Vector	98.48	97.6	98.04
Root Cause	98.68	99.02	98.85
Vulnerability Type	99.03	98.35	98.69
Affected product	98.21	98.36	98.28

is developed atop spaCy [30], we develop baselines utilizing spaCy's `en_core_web_sm` and `en_core_web_lg` encoders [30]. Here, the baselines are chosen because our focus is on comparing the base models, eliminating the differences introduced by the frameworks. We train, fine-tune, and evaluate our NER model and baselines on the same training[1], validation, and test datasets. We evaluate our NER model and baselines with macro metrics, i.e., precision, recall, and F1 score [38].

Evaluation Performance. Table I presents the overall performance of the compared models across all the vulnerability entities. The results show the superior performance of our NER model compared to baselines. Moreover, we also evaluate our NER model on individual vulnerability entities. Table II shows the detailed performance, i.e., precision, recall, and F1 score, of our NER model on the individual vulnerability entities. Among the entities, our NER model performs the best on the "impact" entity but the worst on the "attacker type" entity (see [39]) To understand the performance differences between these two entities, we conducted a manual investigation on our test samples and discovered that vulnerability descriptions often provide more explicit information about the impact rather than the attacker types. As an illustration, the NVD description of CVE-2017-11341 does not mention the attacker type in its description.

B. RQ2: Effectiveness of Our Word Embedding Method

To address **RQ2**, we perform evaluations on our word embedding methods for semantically matching attack graph nodes.

Word Embedding Dataset and Training. Based on the dataset curation framework presented in Figure 3, we are able to obtain 62,544 pre-processed security documents to train our word embedding models, which are stored in the SQLite database. The dataset sampling approach (§III-B) results in 6,335,336 word embedding training samples in total. To model the word semantics in the security context, we choose to train the Continuous Bag-of-Words (CBOW) and Skip-Gram models based on the SentencePiece framework [40]. We train the models to optimal performance by monitoring the loss curve which showed a convergence trend.

Evaluation. To evaluate word embedding, the common practice is using public synonyms datasets, e.g., WS-353 and MTurk-287 [41]. However, to the best of our knowledge, there is no such open dataset in the security context. Additionally, constructing such a dataset with ground truth requires huge human effort and domain-specific expertise, e.g., linguistic and security knowledge. Therefore, we opt to visualize word embeddings by using the t-distributed stochastic neighbor embedding (t-SNE) [42] to evaluate the performance of our word embedding models; t-SNE is an algorithm used for data visualization by reducing high-dimensional data to two or three dimensions [42]. It does so by preserving the local structure of the data and creating a low-dimensional map where similar data points are grouped together. We apply t-SNE to our word embedding models by visualizing the semantic similarity between attack circuit ports, e.g., the input of one attack graph node and the output of another attack graph node. Figure 7 presents an example where the distance of dots representing phrases corresponds to their semantic similarity. In Figure 7, we present the embeddings of the randomly selected phrases identified by our NER models. Note that while we could show embeddings of all phrases, we opt not to do this for clearer visualization. This figure shows that similar semantic phrases are clustered together, such as "cross site scripting flaw" and "XSS vulnerability". The results show that our word embedding models can capture the word semantics well in the security context. Additionally, in [39], we present the computed word embedding scores of each CVE's postconditions against the preconditions of all other CVE samples.

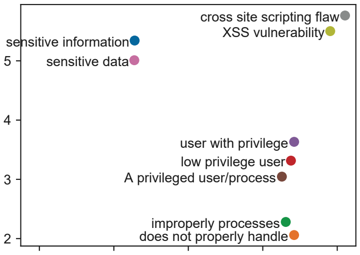


Fig. 7: t-SNE Evaluation of Generated Embeddings. The distance between phrases shows their semantic similarity.

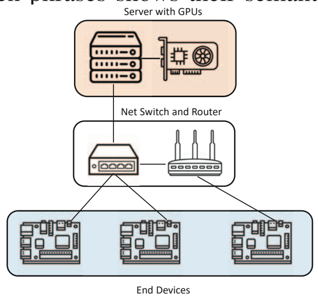
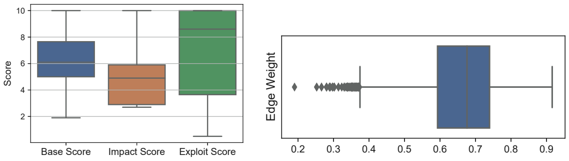


Fig. 8: Test System for GRAPHENE

C. Case Study

To evaluate GRAPHENE on capturing the holistic security posture of computer systems, we test it on the infrastructure shown in Figure 8. The test system includes three major components: (1) Jetson Nano device as the server with GPUs, (2) TP-link devices as network switch and router, and (3) Raspberry Pi boards as end devices. Moreover, we have also adopted the common communication library, OpenSSL, as the software for securing network traffic.

[1] The number of training steps is set to 2000, with which our NER model and baselines exhibited loss convergence.



(a) Vulnerability Scores (b) Edge Weights

Fig. 9: Distributions of Vulnerability Scores and Attack Graph Edge Weights

To assess the security of the test system, we follow the workflow shown in Figure 5. That is, we first curate all 18K CVEs (till June 2022) from the NVD CVE database. We then identify CVEs related to the devices, software and libraries used in Figure 8 by matching the product names in CVE entries. After manually confirming the CVEs, we identified 99 CVEs vulnerability descriptions and reports for the test system, including 4 CVEs for Raspberry Pi, 8 CVEs for Jetson Nano, 35 CVEs for TP-link, and 52 CVEs for OpenSSL.

Next, we collect CVE metadata from the NVD database, i.e., CVE descriptions and vulnerability scores. CVE descriptions carry enriched security information. The mean, median, minimal, and maximum number of words in the descriptions are 48.5, 45, 9, and 131, respectively. Moreover, the most frequent bigram and trigram are "remote attackers" and "denial of service", which appear in the descriptions 38 and 34 times, showing the popular types of attackers and attacks. For risk assessment, Figure 9a presents the distribution of CVE base scores, impact scores, and exploit scores. From this figure, we observe that (1) CVSS base scores show a pretty high risk associated with the CVEs; (2) compared to base scores, the impact scores are in a narrow range; (3) the high impact scores reflect the high likelihood or probability that CVEs will be actively exploited in real-world attacks.

Attack Graph Construction. To identify attack graph nodes and build attack graph edges, we run REST APIs of the named entity recognition (NER) and word embedding models. The average inference time of the NER and word embedding models are 0.023 and 0.0017 seconds for each input, respectively. Afterward, we convert the identified entities into the input, output, precondition, and postcondition for each attack graph node. To build the edges, we calculate the Cosine similar score between attack graph nodes based on the embeddings generated by the word embedding models, which serve as the edge weights. Figure 9b presents the distribution of attack graph edge weights. The distribution shows that the majority of attack graph nodes are with high connectivity, i.e., the average edge weight is 0.683. In our experiments, we applied a threshold of 0.8 for the similarity score. In practical terms, if CVE *A* has a similarity score lower than 0.8 with CVE *B*, there is no edge connecting them in the resulting attack graphs. Our resulting cumulative attack graph contains 100 graph nodes, including one attacker node, 80 CVE nodes, and 19 target (CWE) nodes.

TABLE III: Risk scores for the Architecture of Figure 8

Layer	Commulative	Network	System & HW	ML	Crypto
Exploit score (/10)	2.89	3.57	3.068	5.74	2.94
Impact score (/10)	3.07	3.18	3.8322	5.82	4.277
Risk score (/10)	2.31	2.72	3.9583	5.41	2.61
Total nodes	100	27	30	30	22
Number of attack paths	27297	137	53	6289	20
Shortest attack paths	5	3	4	3	2
Vertex cover size	50	19	15	21	3
Score computation time (seconds)	0.0198	0.0018	0.0014	0.0051	0.00098
Risk analysis time (seconds)	1.1874	0.0053	0.0021	0.2538	0.00097

Risk Scoring. Table III shows some numerical results and performance measures for the scenario in Figure 8. We provide the results for the cumulative attack graph as well as the layered attack graphs. As mentioned in Section IV, the exploit, impact, and risk scores are within the [0-10] range. The score computation time (expressed in seconds) includes the time needed to traverse the generated attack graphs from the graph service and compute the graph exploitability, risk, and impact scores. The risk analysis time (expressed in seconds) is the time needed by the risk scoring system operations, that is, (1) identifying the shortest attack paths and the high-severity paths, (2) identifying key vulnerabilities, and (3) identifying the minimum set of vulnerabilities that cover all the attack paths.

In the cumulative attack graph, GRAPHENE has identified CVE-2020-5215, CVE-2020-15206, and CVE-2021-29540 as the top three critical vulnerabilities associated with the installation of two vulnerable Tensorflow versions (2.4.2 and 1.15.2, respectively) on servers equipped with GPUs. These vulnerabilities are part of the computed vertex cover, indicating their presence in the set of vulnerabilities encompassing all attack paths. Ultimately, GRAPHENE identified the five shortest attack paths within the cumulative attack graph, each having one CVE node with an exploitability score of 10, denoting the highest possible score. These CVE nodes are specifically CVE-2010-3173 (Mozilla Firefox on end devices), CVE-2011-0392 (Cisco TelePresence Recording Server on the server), CVE-2012-6531 (Zend Framework on end devices), CVE-2012-0884 (OpenSSL in all devices and the server), and CVE-2015-0763 (Cisco Unified MeetingPlace on end devices). Notably, analogous observations hold true for the other layers.

VII. DISCUSSION

A key concern with attack graphs generated by our methodology is the potential for inaccuracies, which can impact specific use cases differently. Here, we discuss the effects of these inaccuracies in the context of posture analysis.

Inaccuracies may lead to (1) generating *false paths*—paths that are not feasible—and (2) missing *feasible paths*.

False paths. False paths represent connections between attack nodes that are not feasible in practice. For instance, a multi-step attack requiring phishing, privilege escalation, and lateral movement would be impossible if any intermediate step fails. Despite their infeasibility, these paths remain valuable in GRAPHENE's role as a posture analysis tool, offering insights into potential vulnerabilities and even unlikely attack scenarios. Rather than being flaws, false paths can help refine defense strategies and strengthen security.

Missing paths. Missing paths refer to potential connections between nodes that are absent from GRAPHENE's output, affecting the attack graph's completeness. Although GRAPHENE scans all known vulnerabilities to ensure a comprehensive node-set, it may not generate all possible edges. This could be due to (1) the model failing to infer connections based on pre- and postconditions or (2) insufficient detail in CVE descriptions to establish links. For (1), retraining the model with updated datasets could improve performance, and GRAPHENE allows users to adjust pre- and postconditions to correct errors. For (2), users can identify CVEs with weak or missing correlations through the dashboard and provide additional contextual data, prompting the system to re-process the CVE and potentially generate the missing paths.

The above strategies can be combined with frameworks for automatically assessing the feasibility of attack graphs generated by GRAPHENE or similar systems. One such framework that we envision generates executable attack scripts from the attack graphs using ML applied to publicly available data, such as code snippets and function names. The scripts can then be run in simulated environments, such as Caldera [43], to assess their feasibility.

VIII. RELATED WORK

Attack graph construction. Prior approaches mainly focus on extracting attack information [2], [13], [18], [44]–[49]. They can be categorized as rule-based and learning-based methods. In rule-based methods [2], [13], [45], [46], all rules are manually defined, limiting their scalability. Learning-based methods use traditional machine learning algorithms [18], [47], such as support vector machines, and deep learning models [48], [49]. However, such learning-based methods are usually trained on fuzzy or domain-specific features and thus fail to learn the general vulnerability semantics in the security context. For example, such previous approaches use pre-trained word vectors trained on general English datasets. Therefore they are less accurate in the security domain due to the presence of specific terminology and linguistic semantics.

NER and Word Embedding for Security Applications. The application of NER to the descriptions and analysis of vulnerabilities is not new, and previous works have analyzed vulnerabilities from different aspects with domain-specific NER [18]–[20]. Unfortunately, such approaches have limited scope or use weak machine learning models resulting in a suboptimal performance [50]. Word embedding generates distributed representations of natural language words for efficient computations [27], [51]. Like its applications in natural language processing, there are approaches adopting it for security applications [52]. For example, Shen et al. [52] model the cyberattack steps by temporal word embedding. Srivastava et al. [53] propose enhancing security NLP models by word embeddings. Unlike existing works, we focus on contextualizing the vulnerability-specific word embeddings by training models on massive vulnerability reports, i.e., CVE descriptions.

Risk Scoring. Li et al. [54] proposed a cost/benefit analysis for attack graphs, requiring manual input of cost and benefit values for each node. Lu et al. [7] used graph neural networks and Google's PageRank algorithm to rank the importance of attack graph steps. In contrast, GRAPHENE extends the CVSS scoring system, aligning with the CVE assigner's assessment. Idika et al. [8] focused on attack graph-based metrics, analyzing shortest paths and attack paths without considering severity. Probabilistic approaches, like Liu et al.'s [9] Bayesian network model, require users to assign exploitability probabilities to each CVE, which can lead to inaccuracies and overlook the impact on critical network resources.

IX. CONCLUSION

In this paper, we propose GRAPHENE, a novel automated security posture analyzer to generate attack graphs. Using machine learning, natural language processing, and systematic vulnerability analysis, GRAPHENE offers a comprehensive security solution that can extract semantic meanings from disclosed vulnerabilities, perform layered classification, and create comprehensive attack graphs for holistic security analytics. The multi-layered approach to security posture analysis provided by GRAPHENE has the potential for a more nuanced understanding of a network's security posture, allowing vulnerabilities of different natures and their interrelations to be addressed by subject matter experts. Furthermore, the proposed scoring methods provide a more tailored and relevant security assessment for the infrastructure under analysis, thereby enhancing the practical value of the analytics.

ACKNOWLEDGMENTS

The work reported in this paper has been supported by NSF under grant 2229876 and by Cisco Research.

REFERENCES

[1] X. Jin, C. Katsis, F. Sang, J. Sun, A. Kundu, and R. Kompella, "Edge security: Challenges and issues," *arXiv preprint arXiv:2206.07164*, 2022.

[2] M. U. Aksu, K. Bicakci, M. H. Dilek, A. M. Ozbayoglu, and E. ı. Tatli, "Automated generation of attack graphs using nvd," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 135–142.

[3] B. Bezawada, I. Ray, and K. Tiwary, "Agbuilder: an ai tool for automated attack graph building, analysis, and refinement," in *Data and Applications Security and Privacy XXXIII: 33rd Annual IFIP WG 11.3 Conference, DBSec 2019, Charleston, SC, USA, July 15–17, 2019, Proceedings 33*. Springer, 2019, pp. 23–42.

[4] A. Ibrahim, S. Bozhinoski, and A. Pretschner, "Attack graph generation for microservice architecture," in *Proceedings of the 34th ACM/SIGAPP symposium on applied computing*, 2019, pp. 1235–1242.

[5] Z. Fang, H. Fu, T. Gu, P. Hu, J. Song, T. Jaeger, and P. Mohapatra, "Iota: A framework for analyzing system-level security of iots," in *2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2022, pp. 143–155.

[6] M. Urbanska, I. Ray, A. E. Howe, and M. Roberts, "Structuring a vulnerability description for comprehensive single system security analysis," *Rocky Mountain Celebration of Women in Computing, Fort Collins, CO, USA*, 2012.

[7] L. Lu, R. Safavi-Naini, M. Hagenbuchner, W. Susilo, J. Horton, S. L. Yong, and A. C. Tsoi, "Ranking attack graphs with graph neural networks," in *Information Security Practice and Experience: 5th International Conference, ISPEC 2009 Xi'an, China, April 13-15, 2009 Proceedings 5*. Springer, 2009, pp. 345–359.

[8] N. Idika and B. Bhargava, "Extending attack graph-based security metrics and aggregating their application," *IEEE Transactions on dependable and secure computing*, vol. 9, no. 1, pp. 75–85, 2010.

[9] Y. Liu and H. Man, "Network vulnerability assessment using bayesian networks," in *Data mining, intrusion detection, information assurance, and data networks security 2005*, vol. 5812. SPIE, 2005, pp. 61–71.

[10] S. Abraham and S. Nair, “Cyber security analytics: a stochastic model for security quantification using absorbing markov chains,” *Journal of Communications*, vol. 9, no. 12, pp. 899–907, 2014.

[11] (2024) National vulnerability database. [Online]. Available: https://nvd.nist.gov/

[12] Y. Li and T. Yang, “Word embedding for understanding natural language: a survey,” in *Guide to big data applications*. Springer, 2018, pp. 83–104.

[13] J. Payne, K. Budhraja, and A. Kundu, “How secure is your iot network?” in *2019 IEEE International Congress on Internet of Things (ICIOT)*. IEEE, 2019, pp. 181–188.

[14] Y. Jin, J. Lim, I. Yun, and T. Kim, “Compromising the macOS kernel through Safari by chaining six vulnerabilities,” in *Black Hat USA Briefings (Black Hat USA)*, Las Vegas, NV, Aug. 2020.

[15] X. Feng, X. Liao, X. Wang, H. Wang, Q. Li, K. Yang, H. Zhu, and L. Sun, “Understanding and securing device vulnerabilities through automated bug report analysis,” in *SEC'19: Proceedings of the 28th USENIX Conference on Security Symposium*, 2019.

[16] R. A. Bridges, C. L. Jones, M. D. Iannacone, K. M. Testa, and J. R. Goodall, “Automatic labeling for entity extraction in cyber security,” *arXiv preprint arXiv:1308.4941*, 2013.

[17] X. Jin, K. Pei, J. Y. Won, and Z. Lin, “Symlm: Predicting function names in stripped binaries via context-sensitive execution-aware code embeddings,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1631–1645.

[18] S. Weerawardhana, S. Mukherjee, I. Ray, and A. Howe, “Automated extraction of vulnerability information for home computer security,” in *International Symposium on Foundations and Practice of Security*. Springer, 2014, pp. 356–366.

[19] H. Binyamini, R. Bitton, M. Inokuchi, T. Yagyu, Y. Elovici, and A. Shabtai, “A framework for modeling cyber attack techniques from security vulnerability descriptions,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2574–2583.

[20] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni, “Understanding iot security from a market-scale perspective,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1615–1629.

[21] M. Kuhlmann, C. Gómez-Rodríguez, and G. Satta, “Dynamic programming algorithms for transition-based dependency parsers,” in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011, pp. 673–682.

[22] (2022) Key details phrasing. [Online]. Available: https://cveproject.github.io/docs/content/key-details-phrasing.pdf

[23] M. Naili, A. H. Chaibi, and H. H. B. Ghezala, “Comparative study of word embedding methods in topic segmentation,” *Procedia computer science*, vol. 112, pp. 340–349, 2017.

[24] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[25] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, “Mitre att&ck: Design and philosophy,” in *Technical report*. The MITRE Corporation, 2018.

[26] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, “Ac/c++ code vulnerability dataset with code changes and cve summaries,” in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 508–512.

[27] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[28] FIRST.Org. Common vulnerability scoring system v3.1: Specification document. [Online]. Available: https://www.first.org/cvss/v3.1/specification-document

[29] R. Bar-Yehuda and S. Even, “A local-ratio theorem for approximating the weighted vertex cover problem,” *Annals of Discrete Mathematics*, vol. 25, no. 27-46, p. 50, 1985.

[30] M. Honnibal, I. Montani, S. Van Landeghem, A. Boyd *et al.*, “spacy: Industrial-strength natural language processing in python,” 2020.

[31] R. Rehurek and P. Sojka, “Gensim–python framework for vector space modelling,” *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, vol. 3, no. 2, 2011.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in python,” *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[33] A. Hagberg, P. Swart, and D. S Chult, “Exploring network structure, dynamics, and function using networkx,” Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[34] Flask documentation. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/

[35] Dash documentation. [Online]. Available: https://dash.plotly.com/

[36] J. J. Miller, “Graph database applications and concepts with neo4j,” in *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, vol. 2324, no. 36, 2013.

[37] H. Guo, S. Chen, Z. Xing, X. Li, Y. Bai, and J. Sun, “Detecting and augmenting missing key aspects in vulnerability descriptions,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–27, 2022.

[38] P. Liu, Y. Guo, F. Wang, and G. Li, “Chinese named entity recognition: The state of the art,” *Neurocomputing*, vol. 473, pp. 37–53, 2022.

[39] Graphene. [Online]. Available: https://github.com/graphene-security/graphene-sample-data

[40] T. Kudo and J. Richardson, “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing,” *arXiv preprint arXiv:1808.06226*, 2018.

[41] B. Wang, A. Wang, F. Chen, Y. Wang, and C.-C. J. Kuo, “Evaluating word embedding models: Methods and experimental results,” *APSIPA transactions on signal and information processing*, vol. 8, p. e19, 2019.

[42] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.” *Journal of machine learning research*, vol. 9, no. 11, 2008.

[43] MITRE. Caldera. [Online]. Available: https://caldera.mitre.org/

[44] L. P. Swiler, C. Phillips, and T. Gaylor, “A graph-based network-vulnerability analysis system,” Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 1998.

[45] M. Inokuchi, Y. Ohta, S. Kinoshita, T. Yagyu, O. Stan, R. Bitton, Y. Elovici, and A. Shabtai, “Design procedure of knowledge base for practical attack graph generation,” in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 594–601.

[46] S. Weerawardhana, S. Mukherjee, I. Ray, and A. Howe, “Automated extraction of vulnerability information for home computer security,” in *Foundations and Practice of Security: 7th International Symposium, FPS 2014, Montreal, QC, Canada, November 3-5, 2014. Revised Selected Papers 7*. Springer, 2015, pp. 356–366.

[47] C. L. Jones, R. A. Bridges, K. M. Huffer, and J. R. Goodall, “Towards a relation extraction framework for cyber-security concepts,” in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, 2015, pp. 1–4.

[48] T. Li, Y. Guo, and A. Ju, “A self-attention-based approach for named entity recognition in cybersecurity,” in *2019 15th International Conference on Computational Intelligence and Security (CIS)*. IEEE, 2019, pp. 147–150.

[49] K. Simran, S. Sriram, R. Vinayakumar, and K. Soman, “Deep learning approach for intelligent named entity recognition of cyber security,” in *Advances in Signal Processing and Intelligent Recognition Systems: 5th International Symposium, SIRS 2019, Trivandrum, India, December 18–21, 2019, Revised Selected Papers 5*. Springer, 2020, pp. 163–172.

[50] Y. Yao, J. Duan, K. Xu, Y. Cai, E. Sun, and Y. Zhang, “A survey on large language model (llm) security and privacy: The good, the bad, and the ugly,” *arXiv preprint arXiv:2312.02003*, 2023.

[51] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *Advances in neural information processing systems*, vol. 26, 2013.

[52] Y. Shen and G. Stringhini, “{ATTACK2VEC}: Leveraging temporal word embeddings to understand the evolution of cyberattacks,” in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 905–921.

[53] S. Srivastava, B. Paul, and D. Gupta, “Study of word embeddings for enhanced cyber security named entity recognition,” *Procedia Computer Science*, vol. 218, pp. 449–460, 2023.

[54] W. Li and R. B. Vaughn, “Cluster security research involving the modeling of network exploitations using exploitation graphs,” in *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, vol. 2. IEEE, 2006, pp. 26–26.