AUTOMATED CLASSIFICATION AND MITIGATION OF CYBERSECURITY VULNERABILITIES

by

Ehsan Aghaei

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computer Science

Charlotte

2022

Approved by:

Dr. Xi (Sunshine) Niu

Dr. Bei-Tseng Chu

Dr. Waseem Shadid

Dr. Yasin Raja

ABSTRACT

EHSAN AGHAEI. Automated Classification and Mitigation of Cybersecurity Vulnerabilities. (Under the direction of DR. XI (SUNSHINE) NIU)

With the widespread use of computers and networks today, cybersecurity has emerged as a crucial concern for many businesses as they fight with growing cyber threats by vulnerability exploitation. To identify and mitigate zero-day or unpatched vulnerabilities, intensive defensive measures are required, which calls for a thorough understanding of vulnerability characteristics and threat behavior from several angles. This compels enterprises to spend a considerable amount of money to safeguard their infrastructure from cyberattacks, relying on the costly, ineffective, error-prone, and slow process of experts' input. Therefore, security automation has been a solution for many business owners in the battle against the growing number of cyber threats by vulnerability exploitation.

In recent years, advanced AI technologies in text analytics are gaining wide attention due to their success in a wide range of applications for automating cybersecurity processes. The modern text analytics architectures have been built in novel ways for a variety of applications, assisting cybersecurity professionals in developing resilient mechanisms against threats. Utilizing such technologies can therefore be a viable approach for processing, understanding, and predicting vulnerabilities that are typically reported through unstructured text.

This dissertation utilizes a variety of technologies including deep learning (DL) models, natural language processing (NLP) approaches, and information retrieval (IR) techniques to build a series of models that are able to effectively and efficiently parse, assess, analyze, and mitigate the vulnerabilities based on their textual descriptions reported in Common Vulnerabilities and Exposures (CVE) format. Particularly, it offers a cybersecurity language model, as the core component, which is then utilized

for characterizing the vulnerabilities as well as retrieving the corresponding course of defense actions. As a result of this work, enterprises and cybersecurity researchers will be able to automatically process domain-specific texts, classify vulnerabilities to cybersecurity standards to obtain high-level knowledge, and retrieve the course of defense actions for the underlying threats.

ACKNOWLEDGEMENTS

First and foremost, I want to express my admiration to my father, mother, and two gorgeous sisters for their love and support throughout my life. Thank you for giving me the courage to reach for the stars and pursue my dreams. You don't just mean the world to me, you are my world.

I would like to express my deepest appreciation to my esteemed Ph.D. advisor, Dr. Xi Niu, for her invaluable supervision, support, and tutelage all across my PhD degree. I have benefited greatly from her wealth of knowledge and meticulous supervision in the years of my Ph.D. study. Her unassuming approach to research and outstanding flexibility in communication is a source of inspiration. I hope to carry forward whatever I learnt from her throughout my career.

I could not have undertaken this journey without Dr. Ehab Al-Shaer, whose vast knowledge and wealth of experience have inspired me throughout my academic research and personal life. Dr. Al-Shaer kept in contact with me after he left UNC Charlotte to become a Distinguished Career Professor at Carnegie Mellon University. I'd like to express my appreciation to him for those lengthy brainstorming sessions, technical feedback, and trial-and-error efforts on my proposed models, for which I will be eternally grateful.

Words cannot express my gratitude to Dr. Waseem Shadid for his invaluable assistance in shaping my experiment methods, evaluating my work, and critiquing my results. In addition, I would also thank Dr. Bill Chu and Dr. Yasin Akhtar Raja for their thoughtful comments and recommendations for this dissertation.

Last but not least, I would like to express my sincerest gratitude to Dr. Juile Goodliffe and Sandra Krause not only for their prompt help but also for their kind care. I am immensely thankful to have met such professional and charming people, whose personalities have left an indelible impression on me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: Introduction

With the increasing use of computers and networks in modern day, cybersecurity has emerged as a crucial concern for many businesses as they fight with growing cyberthreats induced by vulnerability exploitation. Vulnerability exploitation consistently results in significant loss of personal and organizational information, as well as billions of dollars for businesses and individuals. Many vulnerabilities remain unpatched for a lengthy period of time. Zero-day vulnerabilities with no available patch, or those that cannot be patched (unpatchable vulnerabilities), require intensive defensive measures to identify, prevent, and/or mitigate, necessitating a thorough understanding of vulnerability characteristics and threat behavior. In the meanwhile, vulnerabilities are required to be classified according to their severity and exploitability, which is critical when prioritizing investigation and defense actions. Hence, measuring the severity of a vulnerability and delivering an impression of how quickly an impacted system should respond to a threat can significantly improve the protection against the threat

According to the CWE/SANS Top 25[1], the primary forms of security vulnerabilities are insecure connections between elements, defective defense, and inefficient resource management. By establishing insecure connections between various systems/networks, an enterprise might become vulnerable to a range of cyber attacks, including SQL injection and cross-site scripting. A defective defense refers to insufficient security methods used to monitor data transfer through a network that does not adequately safeguard the system from adversaries. The term "resource management" refers to the process of allocating, utilizing, producing, and even destroying resources inside a system. A system with insufficient resource management is vulnerable to path traversal and buffer overflow problems.

Corporations invest a significant amount of money each year in safeguarding their

[1]https://www.sans.org/top25-software-errors/

infrastructure from cyberattacks. This operation is commonly labor-intensive and requires expert knowledge, making it expensive, inefficient, error-prone, and slow. Thus, security automation has been a major concern for many business owners in the battle against evolving cyberthreats enabled by attacks. Cybersecurity automation is an ongoing information technology effort that enables organizations and individuals to focus their efforts on more effective defensive measures. While methodologies for data protection have evolved throughout time, the sophistication necessary to assure security has stayed unchanged. Without security automation, analysts must manually handle threats that require investigating and comparing the issue to the threat posed by company information in order to determine its validity, agreeing on a course of action, and then manually resolving the issue, all while dealing with potentially millions of indicators and typically inadequate data.

Modern AI technologies such as deep learning (DL), natural language processing (NLP), and information retrieval (IR) are receiving wide attention due to their effective use in many different tasks as compared to traditional ML models. Recent architectures for text analytics have been modeled in novel ways for a variety of applications that are quite beneficial for cybersecurity professionals striving to provide advanced automation for assessments, minimizing the need for manual work, and leading to millions of dollars in cost savings for industries around the world. Utilizing such technologies can therefore be a viable approach for processing vulnerabilities that are typically reported through unstructured text.

The primary contribution of this dissertation is to automate data assessment and analysis on vulnerabilities, exploits, and, ultimately, effective security measures. It employs a variety of technologies, including deep learning (DL), natural language processing (NLP), and information retrieval (IR), to generate a series of models capable of parsing, assessing, analyzing, and mitigating vulnerabilities based on textual descriptions reported in Common Vulnerabilities and Exposures (CVE) format. This

dissertation, in particular, introduces a cybersecurity language model as the core component, which is then used for characterizing vulnerabilities and retrieving the corresponding course of defense actions, which will enable business organizations and enterprises to automatically process domain-specific texts, classify vulnerabilities to cybersecurity standards to obtain high-level knowledge, and retrieve the course of defense actions for the underlying threats.

1.1 Problem Statement

Malicious users and adversaries are ever racing to exploit newly discovered vulnerabilities before defenders can react. According to FireEye Mandiant Threat Intelligence analysis[2] on vulnerabilities exploited in 2018 and 2019, the bulk of in-the-wild exploitation happens prior to or within a few days following patch deployment, while many other vulnerabilities remain unpatched for a much longer time putting the systems under significant risk. The speed and accuracy with which threats are identified and responded to are two critical factors in cyber defense. According to the Ponemon Institute study[3], the average cost of a data breach is $3.86 million, with a 280-day detection and containment period. Therefore, any approach that can lower such critical factors would be highly beneficial.

Common Vulnerabilities and Exposures[4] (CVEs) are defined as a low-level and product-oriented description of publicly disclosed cybersecurity vulnerabilities which in most cases, they are unable to adequately deliver the characteristic of the threat, including attackers' actions, purposes, and methods. Therefore, CVEs must be defined in a higher abstraction level to expand upon their both threat and defensive utility for vulnerability management, threat hunting, and cyber defense planning.This entails characterizing the CVEs in order to determine the "cause" and "impact" of

[2]https://www.mandiant.com/resources/time-between-disclosure-patch-release-and-vulnerability-exploitation

[3]https://www.ibm.com/security/data-breach

[4]https://cve.mitre.org/

the attack as well as the "methods" leading to the exploit.

There are different standard sources including Common Weakness Enumeration (CWE) and MITRE ATT&CK that offer abstract threat and concrete vulnerability information. The CWE is defined as a hierarchically-designed dictionary of software weaknesses for understanding software flaws, their potential impacts, and identifying means to detect, fix, and prevent the shortcoming [1]. CWEs typically provide knowledge about the system's weaknesses as well as specific information regarding the impact(s) of the associated vulnerabilities if exploited, and a high-level mitigation strategies to minimize the risk. MITRE ATT&CK, on the other hand, is a publicly available knowledge source of adversary tactics and techniques based on real cyber attack observations. These techniques and tactics are organized into matrices by attack stage, ranging from initial system access to data theft or machine control. The MITRE ATT&CK's objective is to compile a thorough list of known adversary tactics and techniques utilized throughout a cyber-attack, encompassing a broad and presumably exhaustive spectrum of attack stages and sequences. MITRE ATT&CK is meant to establish a standardized taxonomy to facilitate communication between organizations. These common security standards are useful for establishing a high-level understanding and for approaching the threat from a variety of angles. In other words, several components of a vulnerability, such as a system weakness, the threat impact, and the techniques used to exploit can be recognized and appraised for effective risk management and subsequently, an efficient and real-time defensive plan.

For example, as illustrated in Fig. 1.1, *CVE-2018-17908* describes a vulnerability in WebAccess/NMS (versions prior to 3.0.2) referring to improper input sanitization that can lead to command injection attack. This CVE provides concrete information about affected products while the associated CWE defines "why" the command injection can exploit by addressing the weakness as improper neutralization of special elements, as well as defining "what" the impacts of exploits are as unauthorized ex-

ecution of codes/commands, denial of service, read/modify files, directories, and/or data, and hide activities. In the meantime, the MITRE ATT&CK defines "how" the attacker can exploit the vulnerability by representing the techniques involved, as well as the abstract goal of the exploit, which is to run malicious code. Such information provides insights commencing with a threat and progressing to a vulnerability or vice versa and has complementary value for assessing the vulnerability and mitigating the underlying threats. Associating CVE reports with MITRE ATT&CK techniques helps vulnerability report authors to establish a clear and uniform approach to defining the impacts and exploitation methods of vulnerabilities for individuals who prepare CVEs, including vulnerability researchers and product vendors. By utilizing ATT&CK, CVEs can convey the tale of what the attacker is attempting to accomplish by exploiting a particular vulnerability. Numerous CVEs focus exclusively on the technical details of exploitation and impact, omitting the malicious actor's higher-level objective. ATT&CK bridges this gap and enables users to comprehend the context of a vulnerability inside an attack scenario and their environment. Utilizing ATT&CK enables consistent reporting of impacts and exploitation methods. While many reporters use the same terminology in their reporting, they commonly used it to describe the same impact differently.

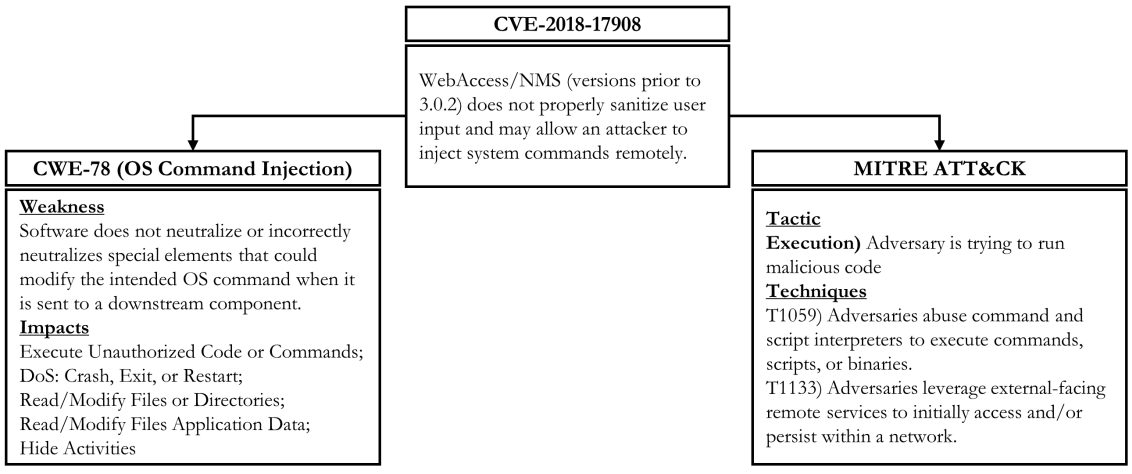


Figure 1.1: An example of associating CVEs to CWEs and MITRE ATT&CK.

In a sense, CWEs and MITRE ATT&CK can assist security analysts in selecting the most effective defense actions to mitigate the threats and safeguard systems. When an attack happens, the threat information contained in the CWE and ATT&CK matrix[5] can help analysts better comprehend the incident and apply the obtained knowledge to improve the security.

In addition, it is crucial to consider that some ATT&CK may not be reachable by CVEs, the mapping is infeasible, the information provided by the technique is limited, or the recommended defense strategy by each standard may not be rich enough to infer the proper countermeasure. CWE and MITRE ATT&CK are primarily used to portray malicious behavior from the adversary's perspective, as well as to provide corresponding defense actions for mitigation and detection. CVEs that are connected to CWEs and ATT&CK techniques enable defenders to promptly analyze the risk associated with a new vulnerability and formulate a mitigation strategy. Such standards offer detection and mitigation methods that can be used to determine whether the mitigations in place are sufficient to address the vulnerability or whether additional mitigations are required. If the defender concludes that additional mitigations are required, they can apply the mappings from MITRE ATT&CK to other resources such as NIST 800-53[6] or CIS Critical Security Controls (CSC)[7] to determine the appropriate actions. The CIS CSC is a recommended set of actions for cyber defense that provide specific and actionable ways to stop today's most pervasive and severe attacks in twenty major security controls. The CSC framework is a set of principles, ideas, etc., that can be used in decisions and judgments (from the MacMillan Dictionary[8]) and it enables the organization, conduct, and management of discussions on security goals and improvements, both within and across communities of enterprises. NIST security controls are the safeguards or countermeasures prescribed for an information

[5]https://attack.mitre.org/matrices/enterprise/
[6]https://ctid.mitre-engenuity.org/our-work/nist-800-53-control-mappings/
[7]https://www.cisecurity.org/controls
[8]https://www.macmillandictionary.com/us

system or an organization to protect the confidentiality, integrity, and availability of the system and its information, components, processes, and data. Therefore, implementing an end-to-end automated framework capable of inferring all possible defense operations against a vulnerability will dramatically improve cyber threat protection in terms of speed, efficiency, and cost, and helpful in efficient defense management.

Fig. 1.2 shows an example of defense actions in terms of mitigation for a given CVE recommended by each standard. It is important to note that such mitigations are different from the patch or a low-level security advisory provided by the vendor. In a nutshell, vulnerability refers to an illness or infection for which there is no vaccination or medication. In this scenario, one must conduct a thorough examination of the cause, symptoms, and impacted areas, and then apply the discoveries to minimize the adverse impact, reduce the risk, and address the underlying causes to avoid it from happening again in the future. In cybersecurity context, the vaccine is a patch that can fix the problem completely, antibiotic is a low-level security advisory that is prescribed by an expert who exactly knows the problem (e.g., vendor) to cure or avoid the exploit until the patch is available. However, as mentioned earlier in many cases patch is unavailable or undesirable and similarly, the vendor security advisory is not released yet. Therefore, to minimize the risk and reduce the impact of the vulnerability, one should take pain relief or antihistamine as mitigation, where mitigations are typically a high-level solution targeting to counter different high-level properties of the threat.

Although investigating entire existing vulnerabilities is important, those commonly exploited and more severe must be the top priority. Hence, automatically ranking the vulnerabilities is required to protect systems against severe threats first. The Common Vulnerability Scoring System (CVSS) is a free and open industry standard for measuring the severity of security vulnerabilities. This provides a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its

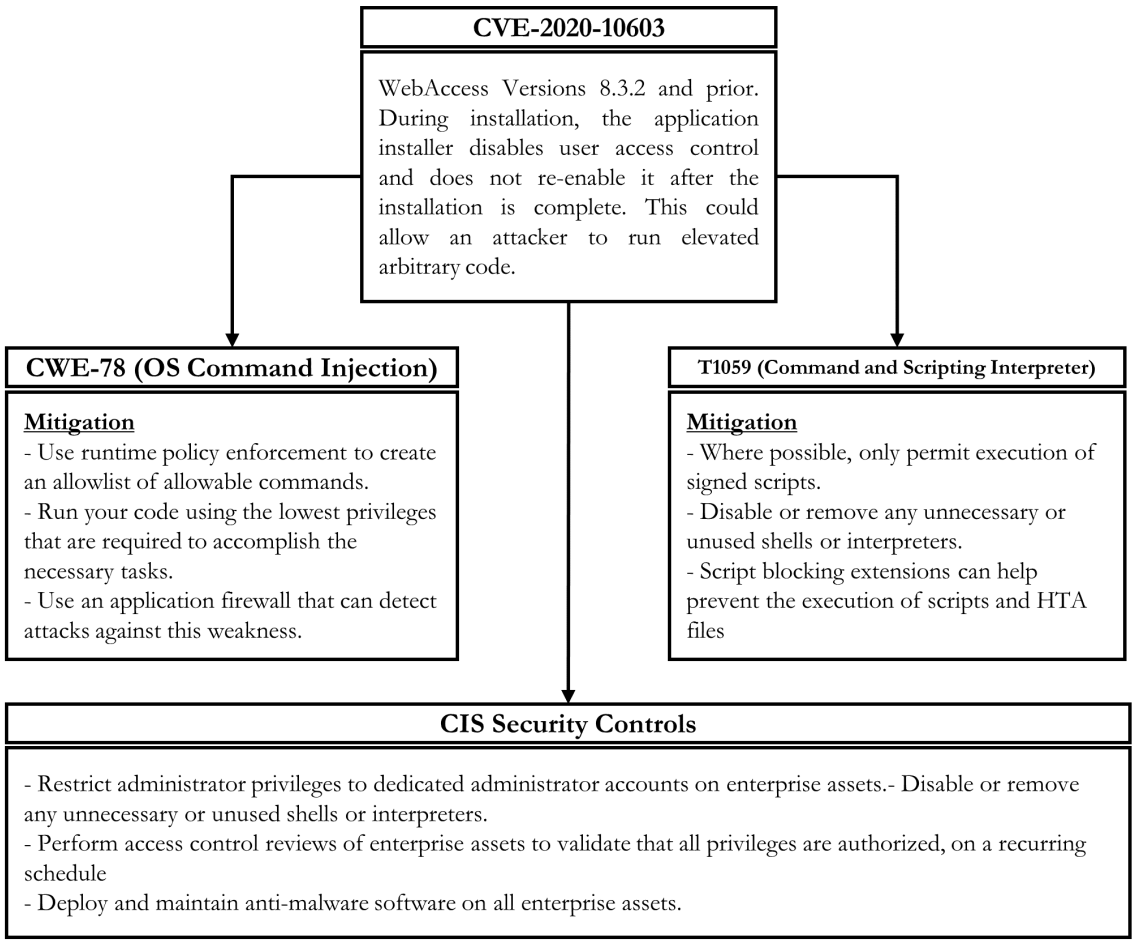


Figure 1.2: An example of associating CVEs to course of actions from CWEs, MITRE ATT&CK, and critical security controls..

severity. The numerical score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management processes.

Currently, the CVSS is calculated through manual engineering effort that is expensive, inefficient, inconsistent, and problematic. The NIST's security experts take days or even longer to analyze CVEs and measure their severities such that many CVEs may remain indeterminate. This controversial cycle simply means cybersecurity analysts cannot rely on the availability of perpetual severity metrics for every CVE. Thus, they are limited to available CVE elements such as description to fo-

cus. Therefore, the only way to overcome this manual prediction challenge is to replicate the expert thinking process and automate the entire scoring method. With this clear goal, automated CVSS score prediction is helpful in CVE characterization from a variety of perspectives. First, it aids in the prioritization of vulnerabilities for threat analysis and risk management. Second, extracting different CVSS factors means obtaining new CVE characteristics that could support in the discovering of an appropriate defense action.

There are two versions of CVSS, V2 and V3 where they are fundamentally different in used metrics and score calculation. Since the newer version introduces several changes in the scoring system that addresses V2's shortcomings reflecting CVEs' characteristics more accurately, in this study, our focus is mainly on V3. CVSS V3 is composed of three metric groups: Base, Temporal, and Environmental. The Base Score reflects the severity of a vulnerability according to its intrinsic characteristics, which are constant over time and assume the reasonable worst-case impact across different deployed environments. The Temporal Metrics adjust the Base severity of a vulnerability based on factors that change over time, such as the availability of exploit code. The Environmental Metrics adjust the Base and Temporal severities to a specific computing environment. They consider factors such as the presence of mitigations in that environment.

Typically, base scores are generated by the organization responsible for the vulnerable product or by a third party scoring on their behalf. It is common to provide just the basic metrics, as these do not vary over time and are applicable to all environments. The base metric group, including exploitability and impact metrics, represents the intrinsic characteristic of CVEs which are everlasting over time and in different environments. The exploitability metrics reflect the ease and technical means by which the CVE can be exploited by presenting the *vulnerable components*. On the other hand, impact metrics present the direct consequence of a successful exploit by

presenting the *impacted components*.

CVEs are issued in the form of text that describes the malicious actions needed to exploit the vulnerability as well as other threat characteristics. Understanding such properties is a master key to successful vulnerability analysis and cyber defense, and automating this process requires a robust text analytic approach capable of capturing context and semantic relationships within the text. Traditional text mining approaches may identify the links between words in textual documents using quantitative analysis or statistics. In the past few decades, these approaches are adequately reasonable for representing text. However, in order to correlate texts with different structures, styles, contexts, and concepts, we need a deep language model that can capture the semantic meaning of the text and its surrounding context. In recent years, NLP researchers used off-the-shelf word embedding models such as Word2Vec [2] and GloVe [3] to initialize the first layer of their neural network model, designed for specific tasks to train in a supervised way over a single dataset. With the rapid growth of unstructured data, emerging technologies such as transfer learning and transformers have brought context understanding to a new level that was not possible before. This dissertation emphasizes the importance of using the state-of-the-art AI technologies such as deep learning and natural language processing to automate this cybersecurity process.

Language modeling is an essential AI application, and the use of neural networks to form language models, known as neural language modeling, has grown in popularity. Neural language modeling (NLM) is a key technology in artificial intelligence and natural language processing (NLP), with applications such as speech recognition, document classification, information retrieval, text production, and machine translation. A language method is a technique for capturing the most important statistical features of the distribution of word sequences in a natural language, often allowing for probabilistic predictions of the next word given the preceding ones. Before neural

networks, traditional language model techniques require creating an n^{th} order Markov chain and estimating n-gram probabilities via counting and subsequent smoothing. Although these statistical models are straightforward to train, the probability of rare n-grams can be underestimated due to data sparsity, even when smoothing techniques are used. NLM address the sparsity of n-gram data by representing words as vectors.

Language modeling is critical in modern NLP applications since it enables machines to represent qualitative text features and convert it to quantitative information, which the machines may then use for the downstream tasks. This technology applies to a wide variety of cybersecurity tasks, including threat hunting, vulnerability analysis and assessment, and cyber threat intelligence. [4, 5].

Given the knowledge of CVEs and associated cybersecurity standards, as well as the availability of modern technologies for an effective and efficient text analytic and classification process, we propose to use these methodologies for processing vulnerability reports in order to characterize, classify, and infer the appropriate course of defense actions. As a result, we develop a cybersecurity language model that represents the cybersecurity-specific language and then fine-tune it for particular cybersecurity tasks in terms of processing vulnerabilities.

1.2 Research Questions

1- Cybersecurity Text Representation

RQ1.1: Does developing a new pre-trained domain-specific language model really help in better representation of cybersecurity text?

2- Cybersecurity Vulnerability Characterization

RQ2.1: Is the cybersecurity specific language model able to prioritize a CVE using CVSS metrics?

RQ2.2: Is the cybersecurity specific language model able to enrich CVE with CWE?

RQ2.3: Is the cybersecurity-specific language model able to detect malicious behaviors in the text?

3- Inferring Course of Action (CoA)

RQ3.1: Is the automated model effective at connecting CVEs to defense actions (CWE mitigation, ATT&CK mitigation, and Critical Security Controls)?

1.3 Research Challenges

Cyber automation has always been a difficult problem due to variety of reasons. In this section, we describe the current challenges in answering the aforementioned research questions.

Semantic Gap

Due to language structure disparities between vulnerability reports and cybersecurity standards, it is difficult to represent vulnerabilities in another structure and view. CVEs are product-specific identifiers that are used to deliver detailed descriptions of vulnerabilities. However, the descriptions of cybersecurity standards are high-level and product agnostic, and each one presents a distinct insight on a threat's property and potential exploitation. On the other hand, similar gaps exist between two reports when one discusses a threat action (e.g., *send a large number of packets*) and the other represents a defense response (e.g., *block the IP address*). Thus, in order to establish the link between these disparate language structures, an effective and efficient semantic analysis is required.

Data Scarcity

While data-driven models such as natural language processing and deep learning require a large amount of data for training, a prevalent issue in cyber analytic problems is a lack of ground truth and labeled data. There are no or a small number of CVEs

that map to standard reference knowledge in this domain. Another major issue in this work is the data imbalance problem. When the incidence of examples belonging to one class is much greater than the incidence of samples belonging to other classes, data imbalancity arises. This may result in inconsistencies in the training process and overfitting. Certain CWEs, for example, are relatively common and have a large number of associated CVEs, whereas others have as few as one sample. Additionally, there are several common CVSS metrics that leave the others with a low frequency.

Lack of Domain-Specific Language Model for Cybersecurity

There are several pre-trained language models for the English language, all of which are made up of general corpora with no domain knowledge. (e.g., word2vec, ELMO, BERT). The drawbacks of domain-agnostic methods are: (1) word representations are general and not tailored for specialized domains; (2) the pre-trained model has only been tested on common English tasks and has never been evaluated on cybersecurity data. For example, BERT and ELMO make the word vector dependent on the word sequence that the word occurs in. While these models reflect the required representation for context, they fail to represent cybersecurity text since it contains many advanced terminologies that are rare or have different meanings in general English. Meanwhile, there is no high-quality off-the-shelf cybersecurity textual dataset to train and evaluate such a model. Lack of data has always been a significant concern in the cybersecurity area, and training a new domain-specific language model requires a massive amount of textual resources, which is not readily available. Therefore, collecting this data and constructing a domain-specific language model is another challenge in this work.

1.4 Contribution to the Knowledge

As mentioned earlier, the main objective of this dissertation is to develop an automated model capable of characterizing and prioritizing vulnerabilities as well as

predicting defense strategies for the underlying attacks based on threat behavior. Our contribution is been divided into three interdependent parts: (1) cybersecurity language model development; (2) CVE characterization; and (3) course of defense action inference.

We begin by crawling and cleaning a large cybersecurity-related corpus from the web and customizing the state-of-the-art language model architecture, namely BERT [6]. This model serves as the foundation for all other parts. In the CVE characterization phase, we provide two different steps "CVE Prioritization" and "CVE Enrichment" frameworks. The former is a stand-alone component for automatically predicting the CVSS metrics of CVEs. In the latter step, we follow the most recent guideline[9] recommended by MITRE to represent the vulnerabilities in higher abstractions using cybersecurity standards. This guideline that is built upon a project defined by Center for Threat-Informed Defense[10], which is a non-profit, privately funded research and development organization operated by MITRE Engenuity. This project establishes a mechanism for assessing the impact of a CVE-listed vulnerabilities using CWEs and MITRE ATT&CK. CWE is a classification of software weaknesses, which are abused by attackers to exploit a vulnerability, rather than a list of specific flaws in products or systems. It returns useful information about the common flaws in the systems and the impact of attacks if exploited. ATT&CK techniques, on the other hand, define how adversaries exploit a vulnerability and what they may accomplish by exploiting the vulnerability. When a vulnerability is described using ATT&CK techniques, it becomes easier for defenders to incorporate it into their threat modeling. The mission is to unify the way vendors, researchers, vulnerability databases, and other sources of vulnerability information describe the impact of vulnerabilities. Defenders can utilize this knowledge to improve the quality of vulnerability management systems, advance the risk models, and enhance the performance of counter-attack approach. The ab-

[9]https://github.com/center-for-threat-informed-defense/attack$_to_cve$/*blob*/*master*/*methodology.md*
[10]https://ctid.mitre-engenuity.org/

stractions derived by CWEs and MITRE ATT&CK, when combined with security control, could help defenders better understand their compensating measures for a particular CVE. Finally, this methodology aspires to establish a key link between vulnerability management and threat modeling in order to come up with an effective course of defense actions.

The guideline defines two categories including "Vulnerability Type" and "Functionality" for CVEs where each category is connected to one or more MITRE ATT&CK technique(s). **Vulnerability type** is defined as a set of similar CWEs that share similar characteristics such as SQL Injection, Unrestricted File Upload, and XML External Entity (XXE). On the other hand, **Functionality** refers to abilities, including obtaining sensitive information, deleting file, or install App, gained by the attacker, which did not have before, as a result of the vulnerability exploitation. We discuss the vulnerability types and functionalities in details in Section 4 and 5, respectively.

We begin by mapping CVEs to CWEs, then to vulnerability types, in accordance with this guideline. Besides that, we link the CVEs to the functionalities that attackers gain access to as a result of the exploit. The CVE classifications mentioned above are not only useful for vulnerability analysis tasks in cyber threat intelligence, but they also provide the researchers with a path to defense actions recommended by CWEs, MITRE ATT&CK techniques, and security controls, which is a crucial step toward creating the required labeled dataset for training an AI model for automating course of action inference. Therefore, we integrate all knowledge gained in the previous step and automatically link CVEs to different set of course of defense actions (mitigations) derived from CWEs, MITRE ATT&CK, and security controls. Fig. 1.3 shows the overview of different phases in this dissertation.

To summarize, we (1) creating a domain-specific language model to represent

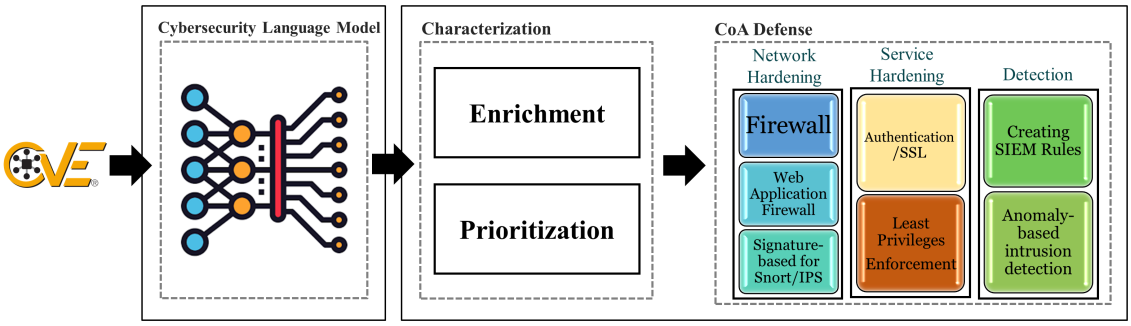


Figure 1.3: Project overview

the cybersecurity language. Then, utilize a reference method for publicly known information-security vulnerabilities and exposures, we characterize the vulnerabilities by enriching CVE reports by (2) prioritizing the CVEs by automatically evaluating the severity score for each vulnerability, leveraging the Common Vulnerability Scoring System (CVSS), and (3) mapping them to CWEs and vulnerability types that are groups of similar CWEs, as well as (4) associating CVEs to threat behaviors and accordingly linking them to a group of MITRE ATT&CK techniques. Finally, (5) we associate vulnerabilities to defense techniques supplied by CWEs, MITRE ATT&CK, and security controls based on the vulnerability characterization. This dissertation contributes to the areas of deep learning, natural language processing, and more importantly cybersecurity by offering a variety of novel predictive model designs to solve important cybersecurity problems.

This dissertations' contributions are summarized as four-fold:

- To our knowledge, SecureBERT is the first cybersecurity-specific language model, which is trained on a large-scale cybersecurity corpus.

- I innovatively applied SecureBERT as a pre-trained model for the important downstreaming cybersecurity tasks through a series of fine-tuning. The downstreaming tasks include characterizing cyber vulnerabilities and retrieving corresponding mitigation methods.

- I collected a large-scale corpus of cybersecurity text for training and evaluating SecureBERT. In addition, using a semi-automatic approach with minimum expert annotations, I made a ground truth dataset for fine-tuning and evaluating a downstreaming vulnerability characterization model, and more importantly for making connections to another downstreaming mitigation identification model. These two datasets are publicly available and can be used by researchers in the cybersecurity community.

- I contributed a unique approach to piece together several important cybersecurity common reference knowledge standards: CWE, CVSS, MIRTRE ATT&CK, CIS Security Controls, and NIST, in order to paint a comprehensive picture of cyber threats from the understanding stage to the mitigation strategies.

CHAPTER 2: A Domain Specific Language Model for Cybersecurity

Natural Language Processing (NLP) has recently gained wide attention in cybersecurity, particularly in Cyber Threat Intelligence (CTI) and cyber automation. Increased connection and automation have revolutionized the world's economic and cultural infrastructures, while they have introduced risks in terms of cyber attacks. CTI is information that helps cybersecurity analysts make intelligent security decisions, that is often delivered in the form of natural language text, which must be transformed into machine-readable format through an automated procedure before it can be used for automated security measures.

This section proposes SecureBERT, a cybersecurity language model capable of representing cybersecurity text (e.g., CTI) and therefore successful in automation for many critical cybersecurity tasks that would otherwise rely on human expertise and time-consuming manual efforts. SecureBERT has been trained using a large corpus of cybersecurity text. It preserves general English representation and more importantly gains comprehension of cybersecurity text through a customized tokenizer as well as an adjustment of word importance. The SecureBERT is evaluated using the standard Masked Language Model (MLM) test as well as two additional standard NLP tasks. Our evaluation studies show that SecureBERT outperforms existing similar models, confirming its capability for solving crucial language understanding tasks in cybersecurity.

2.1 Introduction

The adoption of security automation technologies has grown year after year. The cybersecurity industry is saturated with solutions that protect users from malicious sources, safeguard mission-critical servers, and protect personal information, healthcare data, intellectual property, and sensitive financial data. Enterprises invest on up-to-date technologies to handle such security solutions, typically aggregating a large

amount of data into a single system to facilitate organizing and retrieving key information in order to better identify where they face the risk or where specific traffic originates or terminates. Recently, as social networks and ubiquitous computing have grown in popularity, the overall volume of digital text content has increased. These textual contents span a range of domains, from a simple tweet or news blog article to more sensitive information such as medical records or financial transactions. In cybersecurity context, security analysts process relevant data to detect cyber threat-related information, such as vulnerabilities, in order to monitor, prevent, and control potential threats. For example, cybersecurity agencies such as MITRE, NIST, CERT, and NVD invest millions of dollars in human expertise to analyze, categorize, prioritize, publish and fix disclosed vulnerabilities annually. As the number of products grows, and therefore the number of vulnerabilities increases, it is critical to utilize an automated system capable of identifying vulnerabilities and quickly delivering an effective defense measure.

By enabling machines to swiftly build or synthesize human language, natural language processing (NLP) has been widely employed to automate text analytic operations in a variety of domains including cybersecurity. Language models, as the core component of modern text analytic technologies, play a critical role in NLP applications by enabling computers to interpret qualitative input and transform it into quantitative representations. There are several well-known and well-performing language models, such as ELMO [7], GPT [8], and BERT [6], trained on general English corpora and used for a variety of NLP tasks such as machine translation, named entity recognition, text classification, and semantic analysis. There is continuous discussion in the research community over whether it is beneficial to employ these off-the-shelf models, and then fine-tune them through domain-specific tasks. The assumption is that the fine-tuned models will retain the basic linguistic knowledge in general English and meanwhile develop "advanced" knowledge in the domain while fine tuning [9].

However, certain domains, such as cybersecurity, are indeed highly sensitive to errors, dealing with the processing of critical data and any misconception in this procedure may expose the entire infrastructure to cyber threats, and therefore, automated processing of cybersecurity text requires a robust and reliable framework. Cybersecurity terms are either uncommon in general English (such as *ransomware, API, OAuth, exfiltrate*, and keylogger) or have multiple meanings (homographs) in different domains (e.g., honeypot, patch, handshake, and virus). This existing gap in language structure and semantic contexts complicates text processing and demonstrates the pre-trained English language model may be incapable of accommodating the vocabulary of cybersecurity texts, leading to a restricted or limited comprehension of cybersecurity implications.

In this study, we address this critical cybersecurity problem by proposing a new language model called SecureBERT by employing the state-of-the-art NLP architecture called BERT [6], which is capable of processing texts with cybersecurity implications effectively. SecureBERT is generic enough to be applied in a variety of cybersecurity tasks, such as phishing detection [10], code and malware analysis [11], intrusion detection [12], etc. SecureBERT is a pre-trained cybersecurity language model that can represent both word-level and sentence-level semantics, which is an essential building block for any cybersecurity report. In this context, we collected and processed a large corpus of 1.1 billion words (1.6 million in vocabulary size) from a variety of cybersecurity text resources, including news, reports and textbooks, articles, research papers, and video captions. On top of the pre-trained tokenizer, we developed a customized tokenization method that preserves standard English vocabulary as much as possible while effectively accommodating new tokens with cybersecurity implications. Additionally, we utilized a practical way to optimize the retraining procedure by introducing random noise to the pre-trained weights. To demonstrate SecureBERT's performance in processing both cybersecurity and general English inputs, we conduct

a thorough evaluation using two different tasks: standard Masked Language Model (MLM) and Named Entity Recognition (NER).

2.2 Overview of BERT Language Model

BERT (Bidirectional Encoder Representations from Transformers) [6] is a transformer-based neural network technique for natural language processing pre-training. BERT can train language models based on the entire set of words in a sentence (bidirectional training) rather than the traditional way of training on the ordered sequence of words (left-to-right or combined left-to-right and right-to-left). BERT allows the language model to learn word context based on all surrounding words rather than just the word that immediately precedes or follows it.

BERT leverages Transformers [13], an attention mechanism that can learn contextual relations between words and subwords in a sequence. The Transformer includes two separate mechanisms, an encoder that reads the text inputs and a decoder that generates a prediction for the given task. Since BERTś goal is to generate a language model, only the encoder mechanism is necessary [13]. This transformer encoder reads the entire data at the same time instead of reading the text in order.

Building a BERT model requires two steps: pre-training and fine-tuning. In the pre-training stage, the model is trained on unlabeled data against two different pre-training tasks, namely Masked LM (MLM) and Next Sentence Prediction (NSP). MLM typically masks some percentage of the input tokens (15%) at random and then predicts them through a learning procedure. In this case, the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary. NSP is mainly designed to identifies the relationship between two sentences, which is not directly captured by language modeling. In order to train a model that captures sentence relationships, it trains for a binarized next sentence prediction task that can be trivially generated from any monolingual corpus, in which it takes a pair of sentences as input and in 50% of the times it replaces the second sentence

with a random one from the corpus. To perform fine-tuning, the BERT model is launched with pre-trained parameters and then all parameters are fine-tuned using labeled data from downstream tasks. BERT model has a unified architecture across different tasks, and there is a minor difference between pre-trained and final downstream architecture. The pre-trained BERT model used Books Corpus (800M words) and English Wikipedia (2,500M words) and improved the state-of-the-art for eleven NLP tasks such as getting a GLUE [14] score of 80.4%, which is 7.6% improvement from the previous best results and achieving 93.2% accuracy on Stanford Question Answering Dataset (SQuAD) [15].

A variant of BERT, which is claimed to be a robustly optimized version of BERT with certain modifications in the tokenizer and the network architecture, and ignored NSP task during training, is called RoBERTa [16]. RoBERTa extends BERT's MLM, where it intentionally learns to detect the hidden text part inside otherwise unannotated language samples. With considerably bigger mini-batches and learning rates, RoBERTa changes important hyperparameters in BERT training, enabling it to noticeably improve on the MLM and accordingly the overall performance in all standard fine-tuning tasks. As a result of the enhanced performance and demonstrated efficacy, we develop SecureBERT on top of RoBERTa.

2.3 Data Collection

We collected a large number $(98,411)$ of online cybersecurity-related text files including books, blogs, news, security reports, videos (subtitles), journals and conferences, white papers, tutorials, and survey papers, using our web crawler tool[1]. We created a corpus of 1.1 billion words splitting it into 2.2 million documents each with the average size of 512 words using the Spacy [2] text analytic tool. Table 2.1 shows the resources and the distribution of our collected dataset for pre-training the

[1] `Sampledata:dropbox.com/sh/jg45zvfl7iek12i/AAB7bFghED9GmkO5YxpPLIuma?dl=0`
[2] `https://spacy.io/usage`

SecureBERT.

Table 2.1: The statistics of collected cybersecurity corpora for training the SecureBERT.

Type	No. **Documents**
Articles	8,955
Books	180
Survey Papers	515
Blogs/News	85,953
Wikipedia (cybersecurity)	2,156
Security Reports	518
Videos (subtitles)	134
Total	**98,411**

Vocabulary size	1,674,434 words
Corpus size	1,072,798,637 words
Document size	2,174,621 documents (paragraphs)

This corpora contains various forms of cybersecurity texts, from basic information, news, Wikipedia, and tutorials, to more advanced texts such as CTI, research articles, and threat reports. When aggregated, this collection offers a wealth of domain-specific connotations and implications that is quite useful for training a cybersecurity language model. Table 2.2 lists the web resources from which we obtained our corpus.

2.4 Methodology

In this section, we present two approaches for training the domain-specific language model. We begin by describing a strategy for building a customized tokenizer on top of a pre-trained generic English tokenizer, followed by a practical approach for biasing the training weights to improve weight adjustment and, consequently, a more efficient learning process.

2.4.1 Customized Tokenizer

A word-based tokenizer primarily extracts each word as a unit of analysis, called a token. It assigns each token a unique index, then uses those indices to encode any

Table 2.2: The resources collected for cybersecurity textual data.

Websites
Trendmicro, NakedSecurity, NIST, GovernmentCIO Media, CShub, Threatpost, Techopedia, Portswigger, Security Magazine, Sophos, Reddit, FireEye, SANS, Drizgroup, NETSCOUT, Imperva, DANIEL MIESSLER, Symantec, Kaspersky, PacketStorm, Microsoft, RedHat, Tripwire, Krebs on Security, SecurityFocus, CSO Online, InfoSec Institute, Enisa, MITRE
Security Reports and Whitepapers
APT Notes, VNote, CERT, Cisco Security Reports , Symantec Security Reports
Books, Articles, and Surveys
Tags: cybersecurity, vulnerability, cyber attack, hack
ACM CCS: 2014-2020 , IEEE NDSS (2016-2020), IEEE Oakland (1980-2020) ACM Security and Privacy (1980-2020), Arxiv , Cybersecurity and Hacking books
Videos (YouTube)
Cybersecurity courses, tutorial, and conference presentations

given sequence of tokens. Pre-trained BERT models mainly return the weight of each word according to these indices. Therefore, in order to fully utilize a pre-trained model to train a specialized model, the common token indices must match, either using the indices of the original or the new customized tokenizer.

For building the tokenizer, we employ a byte pair encoding (BPE) [17] method to build a vocabulary of words and subwords from the cybersecurity corpora, as it is proven to have better performance versus a word-based tokenizer. The character-based encoding used in BPE allows for the learning of a small subword vocabulary that can encode any input text without introducing any "unknown" tokens [18]. Our objective is to create a vocabulary that retains the tokens already provided in RoBERTa's tokenizer while also incorporating additional unique cybersecurity-related tokens. In this context, we extract $50,265$ tokens from the cybersecurity corpora to generate the token vocabulary Ψ_{Sec}. We intentionally make the size of Ψ_{Sec} the same with that of the RoBERTa's token vocabulary $\Psi_{RoBERTa}$ as we intended to follow the original RoBERTa's settings as the original works indicated the higher number of

dictionary size does not necessarily lead to better performance.

If Ψ_{Sec} represents the vocabulary set of SecureBERT, and $\Psi_{RoBERTa}$ denotes the vocabulary set of original RoBERTa, both with a size of $50,265$, Ψ_{Sec} shares $32,592$ mutual tokens with $\Psi_{RoBERTa}$ leaving $17,673$ tokens contribute uniquely to cybersecurity corpus, such as *firewall, breach, crack, ransomware, malware, phishing, mysql, kaspersky, obfuscated*, and *vulnerability*, where RoBERTa's tokenizer analyzes those using byte pairs:

$$V_{mutual} = \Psi_{Sec} \cap \Psi_{RoBERTa} \rightarrow 32,592 \texttt{ tokens}$$

$$V_{distinct} = \Psi_{Sec} - \Psi_{RoBERTa} \rightarrow 17,673 \texttt{ tokens}$$

Studies [19] show utilizing complete words (not subwords) for those that are common in a specific domain, can enhance the performance during training since putting subwords back together may be more challenging to understand during model training, as the target word often require attention from multiple subwords. Hence, we choose all mutual terms and assign their original indices, while the remainder of new tokens are assigned random indices with no conflict, where the original indices refer to the indices in RoBERTa's tokenizer, to build our tokenizer. Ultimately, we develop a customized tokenizer with a vocabulary size similar to that of the original model, which includes tokens commonly seen in cybersecurity corpora in addition to cross-domain tokens. Our tokenizer encodes mutual tokens V_{mutual} as original model, ensuring that the model returns the appropriate pre-trained weights, while for new terms $V_{distinct}$ the indices and accordingly the weights would be random.

2.4.2 Weight Adjustments

The RoBERTa model already stores the weights for all the existing tokens in its general English vocabulary. Many tokens such as *email, internet, computer*, and *phone* in general English convey similar meanings as in the cybersecurity domain.

On the other hand, some other homographs such as *adversary, virus, worm, exploit*, and *crack* carry different meanings in different domains. Using the weights from RoBERTa as initial weights for all the tokens, and then re-training against the cyber-security corpus to update those initial weights will in fact not update much leading to overfitting condition in training on such tokens because the size of the training data for RoBERTa (16 GB) is 25 times larger than that for SecureBERT. When a neural network is trained on a small dataset, it may memorize all training samples, resulting in overfitting and poor performance in evaluation. Due to the unbalance or sparse sampling of points in the high-dimensional input space, small datasets may also pose a more difficult mapping task for neural networks to tackle.

One strategy for smoothing the input space and making it simpler to learn is to add noise to the model during training to increase the robustness of the training process and reduces generalization error. Referring to previous works on maintaining robust neural networks [20–22], incorporation of noise to an unstable neural network model with a limited training set can act as a regularizer and help reduce overfitting during the training. It is generally stated that introducing noise to the neural network during training can yield substantial gains in generalization performance in some cases. Previous research has demonstrated that such noise-based training is analogous to a form of regularization in which an additional term is introduced to the error function [23]. This noise can be imposed on either input data or between hidden layers of the deep neural networks. When a model is being trained from scratch, typically noise can be added to the hidden layers at each iteration, whereas in continual learning, it can be introduced to input data to generalize the model and reduce error [24, 25].

For training SecureBERT as a continual learning process, rather than using the initial weights from RoBERTa directly, we introduce a small "noise" to the weights of the initial model for those mutual tokens, in order to bias these tokens to "be a little away" from the original tokens meanings in order to capture their new connotations

in a cybersecurity context, but not "too far away" from standard language since any domain language is still written in English and still carries standard natural language implications. If a token conveys a similar meaning in general English and cybersecurity, the adjusted weight during training will conceptually tend to converge to the original vector space as the initial model. Otherwise, it will deviate to accommodate its new meaning in cybersecurity. For those new words introduced by the cybersecurity corpus, we use the Xavier weight initialization algorithm [26] to assign initial weights.

We instantiated the SecureBERT by utilizing the architecture of the pre-trained RoBERTa-base model, which consists of twelve hidden transformer layers, and one input layer. We adopted the base version (RoBERTa-base) given the efficiency and usefulness. Smaller models are less expensive to train, and the cybersecurity domain has far less diversity of corpora than general language, implying that a compact model would suffice. The model's size is not the only factor to consider; usability is another critical factor to consider when evaluating a model's quality. Since large models are difficult to use and expensive to maintain, it is more convenient and practical to use a smaller and portable architecture.

Each input token is represented by an embedding vector with a dimension of 768 in pre-trained RoBERTa. Our objective is to manipulate these embedding vector representations for each of the $50,265$ tokens in the vocabulary by adding a small symmetric noise. Statistical symmetric noise with a probability density function equal to the normal distribution is known as Gaussian noise. We introduce this noise by applying a random Gaussian function to the weight vectors. Therefore, for any token t, let $\vec{W}_t$ be the embedding vector of token $t$as follows:

$$\vec{W}_t = [w_1^t, w_2^t, ..., w_{768}^t] \tag{2.1}$$

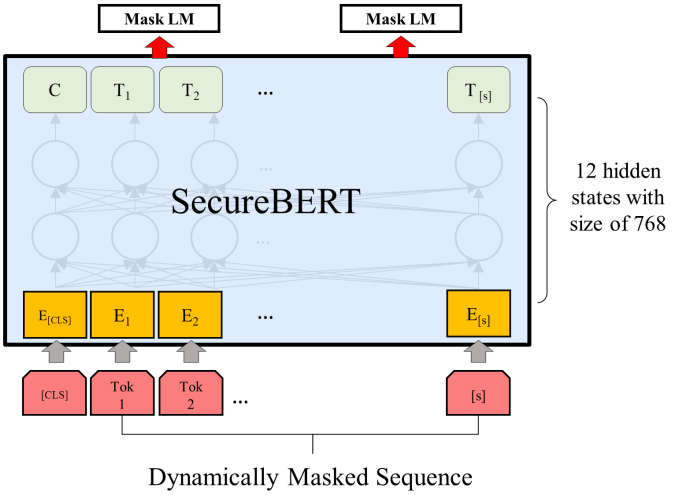


Figure 2.1: SecureBERT architecture for pre-training against masked words.

where w_k^t represents the kth element of the embedding vector for token t.

Let notation $\mathcal{N}(\mu, \sigma)$ be normal distribution where μ denotes the mean and σ the standard deviation. For each weight vector $\vec{W}_t$, the noisy vector $\vec{W}_t'$ is defined as follows:

$$\vec{W}_t' \leftarrow \vec{W}_t \oplus (\vec{W}_t \odot \epsilon), \epsilon \sim \mathcal{N}(\mu, \sigma) \tag{2.2}$$

where ϵ represents the noise value, and $\oplus$ and $\odot$ means element-wise addition and multiplication, respectively.

The SecureBERT model is designed to emulate RoBERTa's architecture, as shown in 2.1. To train SecureBERT for a cybersecurity language model, we use our collected corpora and customized tokenizer. SecureBERT model contains 12 hidden layers and 12 attention heads, where the size of each hidden state has the dimension of 768, and the input word size is 512, and the dimension size is 768. the same as RoBERTa. In RoBERTa ($768 \times 50,265$ elements), the average and variance of the pre-trained embedding weights are -0.0125 and 0.0173, respectively. We picked $mu = 0$ and $sigma = 0.01$ to generate a zero-mean noise value since we want the adjusted weights to be in the same space as the original weights. We replace the original weights in the initial model with the noisy weights calculated using Eq. 2.2.

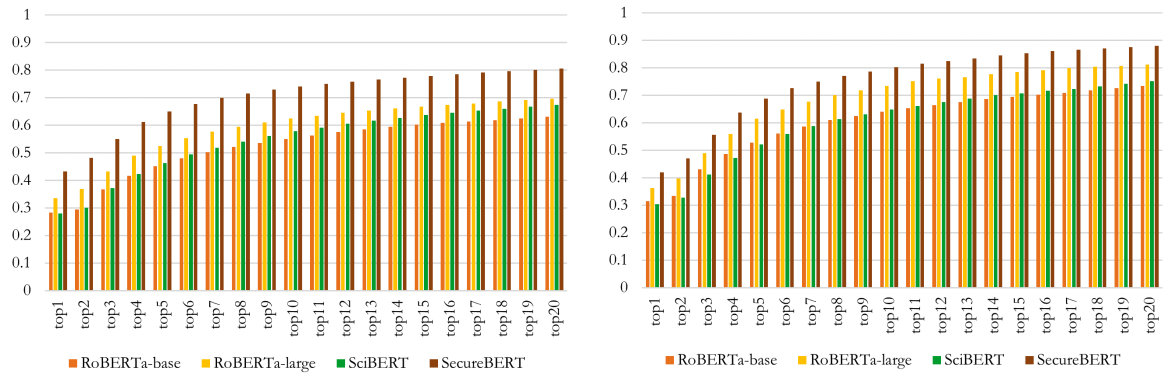
2.5 Masked Language Model Evaluation

We trained the model against MLM task utilizing dynamic masking using RoBERTa's hyperparameters running for $250,000$ training steps for 100 hours on 8 Tesla V100 GPUs with $Batch\_size = 18$, the largest possible mini-batch size for V100 GPUs. We evaluate the model on cybersecurity masked language modeling and other general purpose underlying tasks including named entity recognition (NER) to further show the performance and efficiency of SecureBERT in processing the cybersecurity text as well as reasonable effectiveness in general language.

2.5.1 Masked Language Model (MLM)

In this section, we evaluate the performance of SecureBERT in predicting the masked word in an input sentence, known as the standard Masked Language Model (MLM) task.

Owing to the unavailability of a testing dataset for the MLM task in the cybersecurity domain, we create one. We extracted sentences manually from a high-quality source of cybersecurity reports - MITRE technique descriptions, which are not included in pre-training dataset. Rather than masking an arbitrary word in a sentence, as in RoBERTa, we masked only the verb or noun in the sentence because a verb denotes an action and a noun denotes an object, both of which are important for representing the sentence's semantics in a cybersecurity context. Our testing dataset contains $17,341$ records, with $12,721$ records containing a masked noun ($2,213$ unique nouns) and $4,620$ records containing a masked verb (888 unique masked verbs in total). Figure 2.2a and 2.4b show the MLM performance for predicting the masked nouns and verbs respectively. Both figures present the hit rate of the masked word in $topN$ model prediction. SecureBERT constantly outperforms RoBERTa-base, RoBERTa-large and SciBERT even though the RoBERTa-large is a considerably large model trained on a massive corpora with $355M$ parameters.



(a) Performance in predicting objects.

(b) Performance in predicting verbs.

Figure 2.2: Cybersecurity masked word prediction on RoBERTa-base, RoBERTa-large, SciBERT, and SecureBERT.

Our investigations show that RoBERTa-large (much larger than RoBERTa-base which we used as initial model) is pretty powerful language model in general cybersecurity language. However, when it comes to advance cybersecurity context, it constantly fails to deliver desired output. For example, three cybersecurity sentences are depicted in Fig. 2.3, each with one word masked. Three terms including *reconnaissance, hijacking*, and *DdoS* are commonly used in cybersecurity corpora. SecureBERT is able to represent the context and properly predict these masked words, while RoBERTa's prediction is remarkably different. When it comes to cybersecurity tasks including cyber threat intelligence, vulnerability analysis, and threat action extraction [27, 28], such knowledge is crucial and utilizing a model with SecureBERT's properties would be highly beneficial. The models do marginally better in predicting verbs than nouns, according to the prediction results.

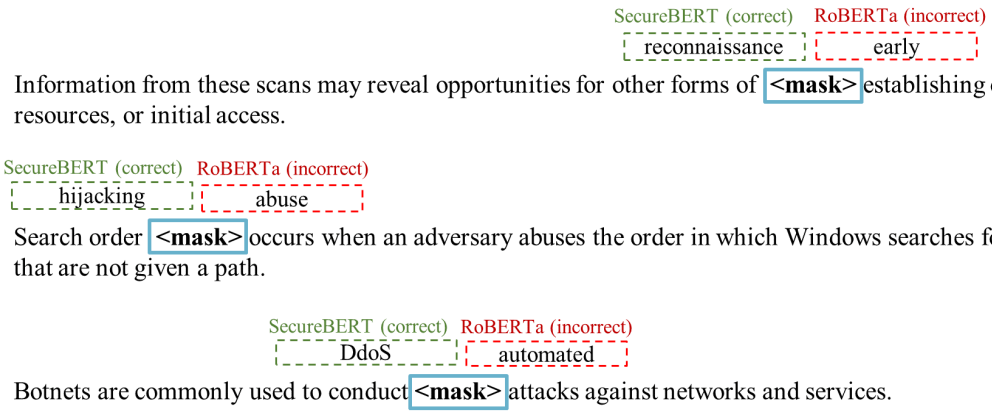


Figure 2.3: A comparative example of predicting masked token. When compared to the off-the-shelf model, RoBERTa-large, SecureBERT demonstrates a better performance in processing the cybersecurity context.

2.5.2 Ablation Study

SecureBERT outperforms existing language models in predicting cybersecurity-related masked tokens in texts, demonstrating its ability to digest and interpret in-domain texts. To enhance its performance and maintain general language understanding, we used specific strategies such as the development of custom tokenizers and weight adjustment.

SecureBERT employs an effective weight modification by introducing a small noise to the initial weights of the pre-trained model when trained on a smaller corpus than off-the-shelf large models, enabling it to better and more efficiently fit the cybersecurity context, particularly in learning homographs and phrases carrying multiple meanings in different domains. As a result of the noise, this technique puts the token in a deviated space, allowing the algorithm to adjust embedding weights more effectively.

In Table 2.3, given a few simple sentences containing common homographs in cybersecurity context, we provide the masked word prediction of four different models, including SB (SecureBERT), SB* (SecureBERT trained without weight adjustment), RB (RoBERTa-base), and RL (RoBERTa-large). For example, word *Virus* in cyber-

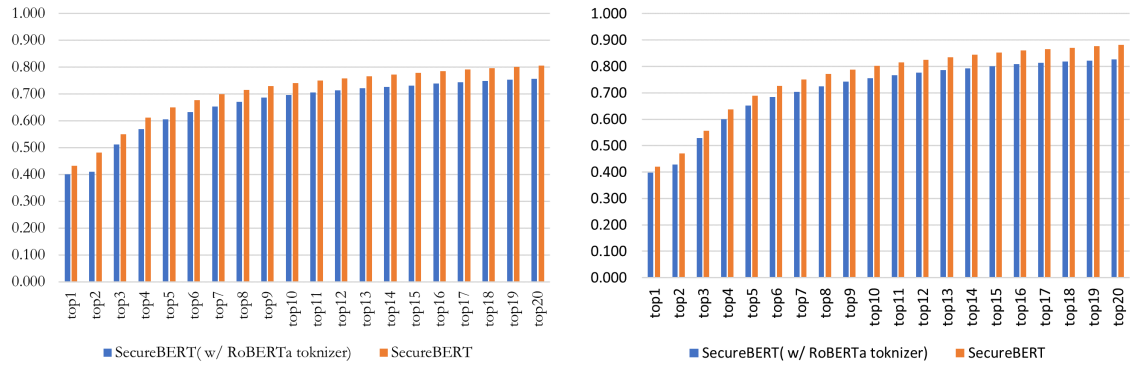
Table 2.3: Shows the masked word prediction results returned by SecureBERT (SB), SecureBERT without weight adjustment (SB*), RoBERTa-base (RB) and RoBERTa-large (RL) in sentences containing homographs

Masked Sentence	Model Predictions
Virus causes <mask>.	**SB**: DoS | crash | reboot **SB***: problems | disaster | crashes **RB**: cancer | autism | paralysis **RL**: cancer | infection | diarrhea
Honeypot is used in <mask>.	**SB**: Metasploit | Windows | Squid **SB***: images | software | cryptography **RB**: cooking | recipes | baking **RL**: cooking | recipes | baking
A worm can <mask> itself to spread.	**SB**: copy | propagate | program **SB***: use | alter | modify **RB**: allow | free | help **RL**: clone | use | manipulate
Firewall is used to <mask>.	**SB**: protect | prevent | detect **SB***: protect | hide | encrypt **RB**: protect | communicate | defend **RL**: protect | block | monitor
zombie is the other name for a <mask>.	**SB**: bot | process | trojan **SB***: worm | computer | program **RB**: robot | clone | virus **RL**: vampire | virus | person

security context refers to a malicious code that spreads between devices to damage, disrupt, or steal data. On the other hand, a *Virus* is also a nanoscopic infectious agent that replicates solely within an organism's live cells. In simple sentence such as "*Virus causes <mask>.*", four models deliver different prediction, each corresponding to associated context. RB and RL return *cancer, infection* and *diarrhea*, that are definitely correct in general (or medical) context, they are wrong in cybersecurity domain though. SB* returns a set of words including *problem, disaster* and *crashes*, which differ from the outcomes of generic models, yet far away from cybersecurity implication. Despite, SB predictions which are *DoS*, *crash*, and *reboot* clearly demonstrate how weight adjustment helps in improved inference of the cybersecurity context by returning the most relevant words for the masked token.

Customized tokenizer, on the other hand, also plays an important role in enhancing the performance of SecureBERT in MLM task, by indexing more cybersecurity related

tokens (specially complete words as mentioned in Section 2.4.1). To further show the impact of SecureBERT tokenizer in returning correct mask word prediction, we train SecureBERT with original RoBERTa's tokenizer without any customization (but with weight adjustment). As depicted in Fig. 2.4a and Fig. 2.4b, when compared to the pre-trained tokenizer, SecureBERT's tokenizer clearly has a higher hit rate, which highlights the significance of creating a domain-specific tokenizer for any domain-specific language model.



(a) Performance in predicting objects.

(b) Performance in predicting verbs.

Figure 2.4: Demonstrating the impact of the customized tokenizer in masked word prediction performance.

Evaluation: Named Entity Recognition In this section, we fine-tune the SecureBERT to conduct cybersecurity-related name entity recognition (NER). NER is a special task in information extraction that focuses on identifying and classifying named entities referenced in unstructured text into predefined entities such as person names, organizations, places, time expressions, etc.

Since general-purpose NER models may not always function well in cybersecurity text, we must employ a domain-specific dataset to train an effective model for this particular field. Training a NER model in cybersecurity is a challenging task since

there is no publicly available domain-specific data and, even if there is, it is unclear how to establish consensus on which classes should be retrieved from the data. Nevertheless, here we aim to fine-tune the SecureBERT on a relatively small-sized dataset that is related to cybersecurity just to show the overall performance and compare it with the existing models. MalwareTextDB [29] is a dataset containing 39 annotated APT reports with a total of 6,819 sentences. In the NER version of this dataset, the sentences are annotated with four different tags including:

Action: referring to an event, such as "registers", "provides" and "is written".

Subject: referring to the initiator of the Action such as "The dropper" and "This module"

Object: referring to the recipient of the Action such as "itself ", "remote persistent access" and "The ransom note"; it also refers to word phrases that provide elaboration on the Action such as "a service", "the attacker" and "disk".

Modifier: referring to the tokens that link to other word phrases that provide elaboration on the Action such as "as" and "to".

In each sentence, in addition, all the words that are not labeled by any of the mentioned tags as well as pad tokens will be assigned by a dummy label ("O") to exclude them in calculating performance metrics.

For Named Entity Recognition, we take the hidden states (the transformer output) of every input token from the last layer of SecureBERT. These tokens are then fed to a fully connected dense layer with N units where N equals the total number of defined entities. Since SecureBERT's tokenizer breaks some words into pieces (Bytes), in

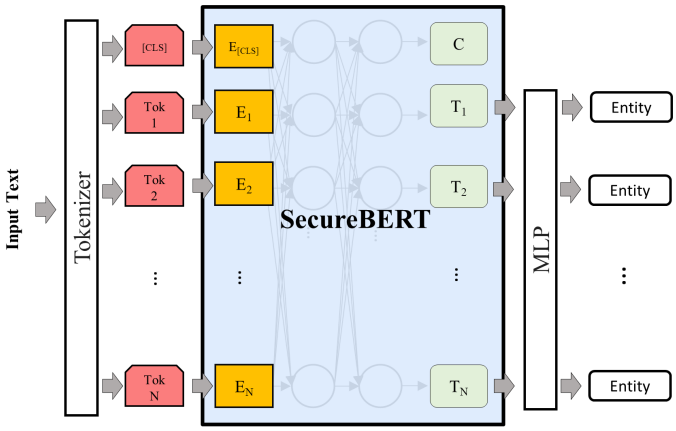


Figure 2.5: SecureBERT architecture for named entity recognition (NER).

such cases we just predict the first piece of the word. Fig. 2.5 shows the architecture of this model.

We trained two versions of the SecureBERT called raw SecureBERT and modified SecureBERT. The former model is the version of our model in which we utilized customized tokenizer and the weight adjustment method, while the latter is the original RoBERTa model trained as is, using the collected cybersecurity corpora. We trained the model for 1,500 steps with $learningrate = 1e-5$ and $Batch\_size = 32$, to minimize the error of $CrossEntropy$ loss function employing $Adam$ optimizer and $Softmax$ as the activation function in the classification layer. Fig. 2.5 shows the SecureBERT's architecture for sentiment analysis.

Table 2.4: the performance of different models trained on MalwareTextDB dataset for NER task.

Model Name	**Precision**	**Recall**	**F1-Score**
RoBERTa-base	84.92	87.53	86.20
SciBERT	83.19	85.84	84.49
SecureBERT (raw)	**86.08**	86.81	86.44
SecureBERT (modified)	85.24	**88.10**	**86.65**

Table 2.4 shows the performance of both SecureBERT's version as well as two other models. Even though the MalwareTextDB dataset contains many sentences with general English implications with limited cybersecurity-specific corpus, modified

SecureBERT outperforms all other models in predicting correct entities.

2.6 Related Works

Beltagy *et al.* [9] unveiled SciBERT following the exact BERT's architecture, a model that improves performance on downstream scientific NLP tasks by exploiting unsupervised pretraining from scratch on a $1.14M$ multi-domain corpus of scientific literature, including 18% computer science and 82% biomedical domain documents.

In a similar work, Gu *et al.* [30] introduced BioBERT focusing particularly on the biomedical domain using BERT architecture and publicly available biomedical datasets. This work also creates a benchmark for biomedical NLP featuring a diverse set of tasks such as named entity recognition, relation extraction, document classification, and question answering. ClinicalBERT [31] is another domain adaptation model based on BERT which is trained on clinical text from the MIMIC-III database.

Thus far, utilizing language models such as BERT for cybersecurity applications is quite limited. CyBERT [32] presents a classifier for cybersecurity feature claims by fine-tuning a pre-trained BERT language model to identify cybersecurity claims from a large pool of sequences in ICS device documents. There are also some other studies working on fine-tuning BERT in the cybersecurity domain. Das *et al.* [33] fine-tunes BERT to hierarchically classify cybersecurity vulnerabilities to weaknesses. Additionally, there are several studies on fine-tuning BERT for NER tasks such as [34], [35] and [36]. Yin *et al.* [37] fine-tuned pre-trained BERT against cybersecurity text and developed a classification layer on top of their model, ExBERT, to extract sentence-level semantic features and predict the exploitability of vulnerabilities. There is also another model called SecBERT [3] published in Github repository which trains BERT on cybersecurity corpus from "APTnotes"[4], "Stucco-Data: cybersecurity data

[3] `https://github.com/jackaduma/SecBERT`
[4] `https://github.com/kbandla/APTnotes`

sources"[5], "CASIE: Extracting Cybersecurity Event Information from Text"[6], and "SemEval-2018 Task 8: Semantic Extraction from CybersecUrity REports using Natural Language Processing (SecureNLP)"[7]. However, at the time of this dissertation, we could not find any article presenting this work to learn more about the details and the proof-of-concept to discuss.

2.7 Conclusions and Future Works

This study introduces SecureBERT, a transformer-based language model for processing cybersecurity text language based on RoBERTa. We presented two practical ways for developing a successful model that can capture contextual relationships and semantic meanings in cybersecurity text by designing a customized tokenization tool on top of RoBERTa's tokenizer and altering the pre-trained weights. SecureBERT is trained to utilize a corpus of 1.1 billion words collected from a range of online cybersecurity resources. SecureBERT has been evaluated using the standard Masked Language Model (MLM) as well as the named entity recognition (NER) task. The evaluation outcomes demonstrated promising results in grasping cybersecurity language.

[5] https://stucco.github.io/data/
[6] https://ebiquity.umbc.edu/_file_directory_/papers/943.pdf
[7] https://ebiquity.umbc.edu/_file_directory_/papers/943.pdf

CHAPTER 3: Automated CVSS Prediction

The previous chapter introduced a new pre-trained cybersecurity language model, a.k.a SecureBERT, that delivered a competitive performance in processing cybersecurity texts. This chapter proposes an innovative methodology to fine-tune SecureBERT to process CVE descriptions and predict the corresponding CVSS metric values for the purpose of vulnerability prioritization.

3.1 Introduction

The Common Vulnerability Scoring System is a free and open industry standard for measuring the severity of security vulnerabilities. This provides a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. The numerical score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management processes.

Currently, the CVSS is calculated through manual engineering effort that is expensive, inefficient, inconsistent, and problematic. The NIST's security experts take days or even longer to analyze CVEs and measure their severities such that many CVEs may remain undecided for a long period of time. Such a controversial cycle implies that cybersecurity analysts cannot rely on the availability of perpetual severity metrics for all CVEs and therefore, they must focus on available CVE elements such as descriptions to obtain crucial CVE severity measures. This controversial cycle simply means cybersecurity analysts cannot rely on the availability of perpetual severity metrics for every CVE. Thus, they are limited to available CVE elements such as description to focus.

Therefore, mimicking expert thinking and automating the entire scoring procedure is the ultimate solution to tackle this manual prediction problem. With this aim in mind, the automatic prediction of CVSS score is beneficial in CVE characterization

Table 3.1: Shows different vectors required to represent CVSS V3.

CVSS V3		
Base Vector	**Temporal Vector**	**Environmental Vector**
Attack Vector (AV)		
Attack Complexity (AC)		Modified Base Metrics (M*)
Privileges Required (PR)	Exploit Code Maturity (E)	Confidentiality
User Interaction (UI)	Remediation Level (RL)	Requirement (CR)
Confidentiality (C)	Report Confidence (RC)	Integrity Requirement (IR)
Availability (A)		Availability Requirement (AR)
Integrity (I)		
Scope (S)		

from two points of view. First, it helps to prioritize vulnerabilities for threat analysis and risk management. Second, extracting different CVSS factors means obtaining new characteristics of CVEs which might help discover an appropriate defense action.

When it comes to properly document the features and severities of software vulnerabilities, CVSS is the standard protocol, which has been issued in two versions, with CVSS V2, and CVSS V3 (and CVSS V3.1) [1] both being extensively utilized in the computer security domain. NVD analysts assign CVSS scores by evaluating the CVEs using specific instructions. Both CVSS versions are structured similarly, providing a series of multiple-choice questions to the vendors concerning the vulnerability's properties.

In general, CVSS is classified into three vectors including base, temporal, and environmental. The base vector defines the inherent properties of the vulnerability, the temporal vector defines how those properties vary over time, and the environmental vector illustrates the severity of a vulnerability in the context of a particular organization. We will concentrate on CVSS V3 in this work as it is more recent and broadly applied recently than the former version.

Table 3.1 shows the different vectors required by CVSS V3. When the security analysts have calculated all of the vectors and values for the underlying metrics, they are integrated using a common syntax to generate a CVSS Vector. Each CVSS version

[1] `https://www.first.org/cvss/`

Table 3.2: Shows the CVSS V3 Base metrics and the potential values.

CVSS V3 Base Metric	
Metric	**Values (classes)**
Attack Vector (AV)	Network (N) | Adjacent (A) | Local (L) | Physical (P)
Attack Complexity (AC)	Low (L) | High (H)
Privileges Required (PR)	None (N) | Low (L) | High (H)
User Interaction (UI)	None (N) | Required (R)
Confidentiality (C)	None (N) | Low (L) | High (H)
Availability (A)	None (N) | Low (L) | High (H)
Integrity (I)	None (N) | Low (L) | High (H)
Scope (S)	Unchanged (U) | Changed (C)

has a non-linear severity formula that takes all the metric values from each vector as input and returns a severity score between 0.0 and 10.0.

Since this equation is not linear, slight changes in the vector can result in significant variances in the severity score. CVSS V3 and V3.1 have the same vectors and differ solely in some metrics and accordingly severity calculation algorithm, which affects a small number of vulnerabilities. In the rest of our work we use the term "CVSS V3" to describe both the CVSS V3 and V3.1 specifications. In addition, since Temporal and Environmental vectors are derivable from Base Vector, we only process the Base vector, which consists of eight different metrics including *Attack Vector (AV)*, *Attack Complexity (AC)*, *Privileges Required (PR)*, *User Interaction (UI)*, *Confidentiality (C)*, *Availability (A)*, *Integrity (I)*, and *Scope (S)*. Each metric also can get different values which is demonstrated in Table 3.2. Our target in this work is to automatically predict the value of each eight metrics for a given vulnerability using the corresponding CVE ID description, utilizing the pretrained SecureBERT language model.

For security analysts such as NIST's, it typically takes days or weeks to assess and annotate vulnerabilities resulting in a huge number of unprocessed vulnerabilities which have been publicly disclosed for a long time. Therefore, relying solely on human expertise provided by databases such as NVD to analyze and prioritize vulnerabilities is not practical and researchers must focus on the early available piece of data at the

time of disclosure such as human-readable descriptions and possibly public references like vendor reports.

Such a manual procedure is costly, and a real-time threat evaluation of new vulnerabilities necessitates a large amount of labor due to the growing number of disclosed vulnerabilities every day; thus, it would be more affordable to organizations if it could be automated, which consequently allows timely manner and intuitive defense planning including security policy reconfiguration, limiting access, and/or shutting them down while waiting for remediation to be applied, all based on the severity of the vulnerability. In this chapter, we propose an automated system that uses the free-form text descriptions of disclosed vulnerabilities to predict the CVSS base vector of these vulnerabilities and then we discuss the shortages of automating CVSS prediction and the limitation of the proposed model.

3.2 Related Works

To the best of our knowledge, there are only a few efforts in automating CVSS prediction in recent years. Khazaei *et al* [38] introduced an objective method for CVSS V2 score calculation by extracting textual features from CVE descriptions and employing SVM, Random Forest, and fuzzy systems for prediction. In this study, the descriptions are tokenized, and after performing standard text mining preprocessing such as filtering the stop-words and stemming, the TF-IDF score of words is calculated, and the dimension is reduced using PCA and LDA techniques. The CVSS V2 scores are roughly predicted within the interval of $[i, i+1)$ where $i \in \{0, 1, 2, ..., 9\}$. This works reported 88.37% accuracy through testing the fuzzy system implementation. This work leveraged an off-the-shelf machine learning model and failed to consider and report many important metrics in CVSS prediction. This work exclusively focused on finding the direct relationship between CVE description and the final score, while original scores are calculated based on different CVSS metric values where each metric specifies a variable in the non-linear score calculation formula.

This direct method performs as a classification task which is problematic since one word (feature) might have a different impact on each metric, and training every metric in one shot is error-prone. In the meantime, the bag-of-word approach does not take the context and semantics into account in to-be-announced CVE reports, such as synonym words and abbreviations. In addition, there is no evidence showing the performance of this tool in predicting different value classes since different classes have a different number of samples, and predicting those who have less number of samples is important.

In a similar approach, Elbaz *et al.* [39] implemented another technique based on linear regression to automatically predict the CVSS vector of newly disclosed vulnerabilities using their textual description. They used a similar bag-of-word approach and represented each CVE by word frequency vector. After filtering stop-words, two different dimension reduction approaches were applied to select the most predictive features from the corpus. In the first approach, the software vendor, product name, and software target existing in the Common Platform Enumeration (CPE), and all the words in CWE titles are collected, and a white-list is generated to find and discard irrelevant features. In the second approach, the conditional entropy score for each word associated with each CVSS metric is calculated, and the top N words with the lowest score are selected for evaluation. Then a linear regression model is created to predict the scores for each CVSS metric. The accuracy of predicting each metric is varied from 60% to 95%. This study suffers from certain weaknesses. First, similar to the previous work, the features are limited to the keywords extracted from CVE reports and CWE titles. This corpus fails to cover context and accordingly, semantic features such as synonyms, abbreviations, and similar words. Hence, there might be some less frequent words uniformly distributed among different values of one metric with a strong relationship with other words, which is a key signature for one specific value. The addressed feature extraction method in this study cannot find such a

relationship, and therefore, it will assign a high entropy score to this word, and it will automatically be ignored in the classification. In addition, a limited corpus without semantic feature analysis may not handle a new word or different writing standard, which is very likely to have since different vendors have different writing styles and new concepts might be raised in the future CVE description. In addition, the evaluation is incomplete and failed to address some important points. About half of the metrics are imbalanced, which means that the frequency of instances for one metric is much higher than the others. E.g, in *Attack Vector*, there are $\sim 35K$ instances with *Network* value, while there are less than 500 samples with *Physical* value out of $28,000$ existing labeled CVEs. This can easily lead to poor predictive performance, specifically for the minority class during training, which has not been addressed in this study.

3.3 Challenges

Automating CVSS vector prediction is a challenging task. The existing works are limited or built upon imprecise assumptions. Predicting the CVSS score requires a deep analysis of each element in the vectors and therefore, a direct approach to obtain the final severity score would be problematic since any small mistake may compromise the entire output significantly. Instead, dividing the CVSS into smaller elements and approaching the problem gradually would provide multi-level information about the vulnerability (each metric provides unique threat behavior). This would become even more important since it is proven that vulnerability descriptions often do not provide enough information about the entire threat characteristics, so any partial information about the threat would be beneficial in case the automated model cannot provide exact values for each metric of the CVSS vector.

On the other hand, traditional information extraction methods and machine learning algorithms may not be the most effective approach to tackle this problem. Such methods mainly work based on statistical analysis and word frequency, treating the

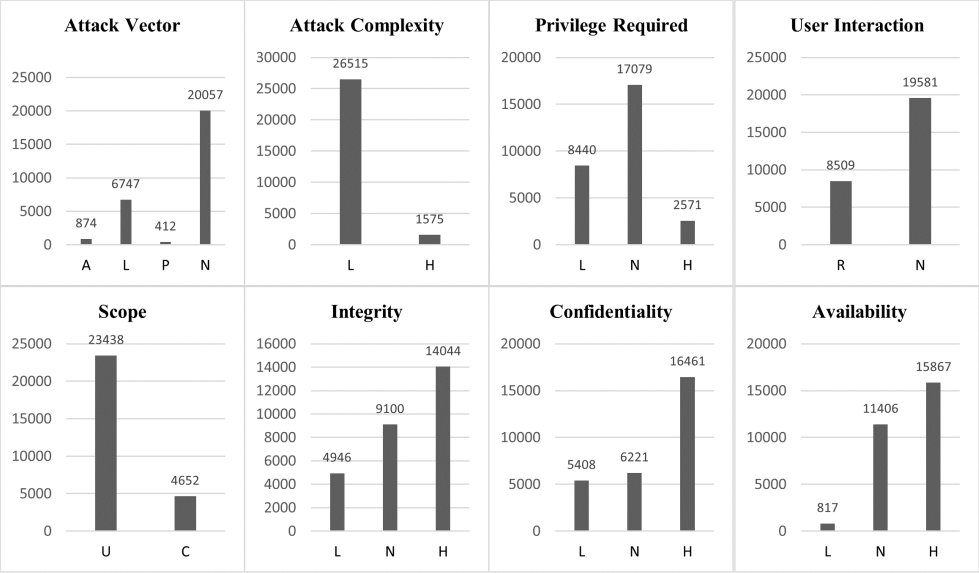


Figure 3.1: Numbers of available labeled records in each value for each CVSS metric. This shows the imbalance data problem in CVSS dataset

available text as a bag-of-words. This strategy is unable to adequately capture the context and extract semantic relationships throughout the text, leading to missing important information and consequently a non-robust predictive model.

The training dataset comprises CVE descriptions whose CVSS vector is already assigned by NVD. In other words, each CVE in this dataset contains a CVSS base metric vector. This dataset is highly unbalanced, which implies that the frequency distribution of a specific value within each metric may be highly different than the others. Existing works failed to adequately address the imbalanced data problem and did not provide a detailed performance evaluation regarding this issue in the CVSS dataset. Table 3.1 shows the value distribution within each metric.

3.4 Model Design

This section describes the CVSS vector prediction pipeline. Since there are eight metrics, we train eight individual models, where the input is the CVE description of a vulnerability and the output is the predicted value for the target metric. A high-level overview of the proposed CVSS analysis pipeline is depicted in Fig. 3.2. We utilized

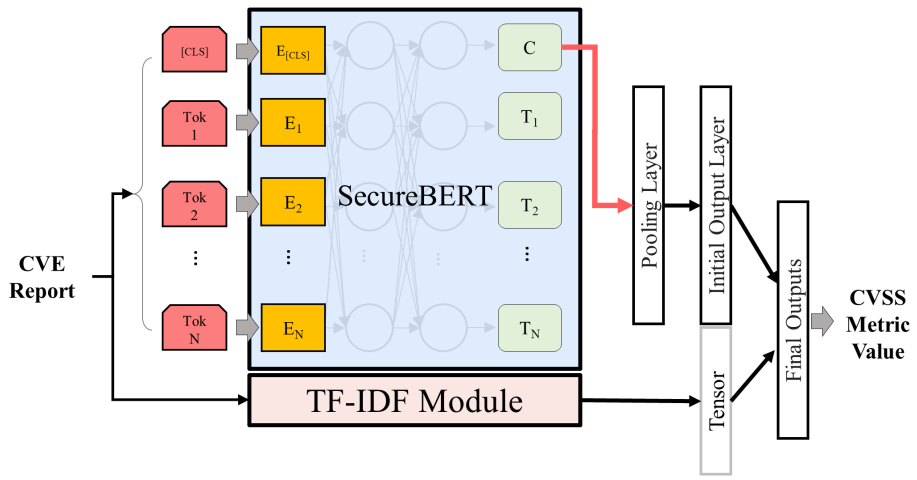


Figure 3.2: CVSS metric value prediction model design. For each CVSS metric, there is a separate model that is trained independently.

the pre-trained SecureBERT as the base model and added an extra TF-IDF module to help the model process the imbalance data. In this architecture, the model takes the CVE description, tokenizes it using SecureBERT's BPE tokenizer, and passes it through the transformer layers. Then, the $[CLS]$ token vector representation is connected to another same-size fully connected layer. This layer, known as pooling layer, takes the aggregated representation of the entire input and passes it through another fully connected layer to return the initial output layer with a size equal to the number of values in a metric. Afterwards, the initial output vector is merged with another vector (tensor) that represents the customized TF-IDF vector from the input CVE description. Finally, the merged vector is connected to the final dense layer (feed-forward neural network) in such a way that its size equals the number of potential values in the metric of interest.

TF-IDF Module

The TF-IDF which stands for term frequency-inverse document frequency is a statistical approach that quantifies the semantic importance of a word in a collection of documents. TF-IDF helps to calculate numeric values for each word in a document corresponding to its semantic importance to that document. TF-IDF is an effective

method in text mining to identify words that contribute much in a particular document and rarely or never contribute in other documents [40], whose occurrences strongly represent a particular category, context, or class, as known as signature words.

As mentioned earlier, CVSS data is highly unbalanced, and to reduce the impact of this imbalancity during the training, we utilize a customized TF-IDF module that generates word-frequency-based vectors, representing the input data concerning the minority classes (values) in models corresponding to each metric. In a nutshell, our studies show that when we consider each potential value within each metric as a class, there are some words, known as signature words, that frequently appear in minority classes. For example, in CVEs whose the value of *Attack Vector (AV)* metric is *Adjacent Network (A)*, words like "adjacent", "Bluetooth", "pairing", and "dongle" are frequently appeared in the description. In another example, when the value of *Attack Complexity (AC)* is *High (H)*, "man-in-the-middle", "memory-cache", "padding", and "prolog" are frequently observed. On the other hand, keyword matching for predicting the value would not be an effective approach as these words do not uniquely appear in one specific class. In addition, when the training data is not large enough, even a powerful language model will not be able to learn the context and semantic relationships sufficiently.

Here, we aim to provide an external hint to the model to help recognize the minority classes during the training using TF-IDF scores, to extract signature words and convert them to a vector to merge with SecureBERT's output. We assume that an extra emphasis on important yet infrequent signature words during the training would benefit the model to adjust its weights to better recognize minority class(es).

In this context, we first conduct the standard text cleaning steps on the training dataset such as removing stop words, special characters, punctuations, and numbers, performing stemming, and then within each metric, tokenize all reports belonging

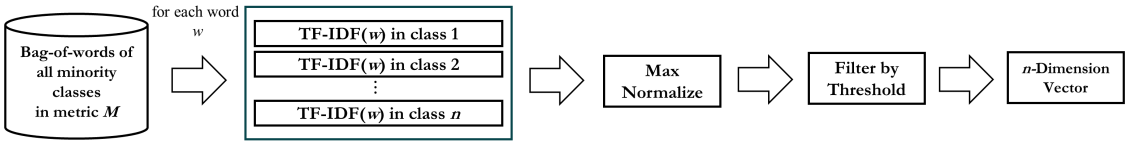


Figure 3.3: Shows the different steps in generating customized TF-IDF vectors to represent CVE descriptions.

to minority class and create a dictionary of words. Then, we calculate the TF-IDF score of each word in this dictionary against all classes. In other words, if metric M contains n different classes, every words in the dictionary is represented by n TF-IDF score, each corresponds to an individual class in a vector of size n. This vector then is normalized to re-scale the scores to a range of 0 to 1. These normalized values represent the statistical relevance of a term belonging to a class, with values closer to 1 indicating higher importance. Since we aim to only identify minority classes using TF-IDF score vectors, we implement a filtering procedure based on a hyperparameter threshold to select words whose scores in minority class(es) are "significantly" higher than their scores in majority classes. As a result, every vulnerability description is defined by a set of n-dimension TF-IDF vector if it shares any word with minority class word dictionary, where n refers to the number of existing classes in metric M. Then, we take the average of these vectors as the final TF-IDF vector representation of the input description. Fig. 3.3 shows the high level overview of TF-IDF vector creation.

Let $\delta(M)$ be the dictionary of terms in minority class(es) in metric M:

$$\delta(M) = \{w_1, w_2, ..., w_r\} \tag{3.1}$$

If $M = \{c_1, ..., c_n\}$ represents the existing classes in metric M, for each $w \in \delta(M)$, we return n scores (TF-IDF scores) corresponding to each $c \in M$:

$$T(w, M) = [t^w_{c1}, ..., t^w_{cn}]. \tag{3.2}$$

Then we normalize each vector $T(w_i, M)$ by dividing all elements by the maximum value of the vector as $MAX[T(w_i, M)] + \epsilon$ (ϵ is a small number added to avoid divide by zero) and call it $T'(w_i, M)$.

Here, our target is to provide a secondary representation of the input text in addition to SecureBERT's output, by conducting a statistical analysis to find those terms which are highly likely to appear in minor classes and rare in other classes. This vector representation will be concatenated by the SecureBERT's output and yield to the classification layer together. Therefore, given $T'(w, M)$, we first define a probability threshold $0 < th < 1$. Then, for each $T'(w_i, M)$, make sure its score corresponding to one potential value within a particular metric is greater than th ($t^w_{i,fj} > th$) and the score corresponding to the other values are smaller than $1 - th$, otherwise remove the word from the dictionary. In other words, we compare all n number of TF-IDF scores associated with each word with the threshold and identify those whose value corresponding to a minority class is significantly higher than other classes (according to the threshold). Any word that satisfies both conditions will be added to the signature term dictionary $Dict(M)$ corresponding to each metric. Therefore, $Dict(M)$ contains words which are highly informative about minority values within each metric.

For any training input CVE description within each metric prediction model, we check how many terms it shares with $Dict(M)$. Let $S = \{w'_1, ..., w'_n\}$ represent the terms in a CVE description, and $g(S, M) = S \cap Dict(M)$ be the set of shared terms between S and $Dict(M)$. For any term $w' \in g(S, M)$, we retrieve the TF-IDF vectors using Eq. 3.3 and take the weighted average ($\overline{\mu}$) of all vectors to construct the TF-IDF representation of the CVE:

$$\mathcal{V}(S, M) = \overline{\mu} \ \ (T'(w', M)), \forall w' \in g(S) \tag{3.3}$$

$\mathcal{V}(S, M)$ is a vector with a size equal to the number of values within a metric

representing the probability score of a CVE being associated with each value, with respect to the signature words. Since signature words are extracted from minor classes, the vector elements associated with the dominant class samples are quite often zero or close to zero, indicating that this vector represents the likelihood that a CVE is associated with minor value, if the CVE shares any words with the signature word dictionary $Dict(M)$; otherwise, all elements would be zero. Taking the weighted average is preferable to taking the maximum or minimum in generating representative vectors. As we aim to maximize the value of the minority class in calculating CVE's TF-IDF representation, taking the minimum would not be applicable. On the other hand, the signature words are not unique implying that such words can occur (even rarely) in other documents including those associated with dominating classes. In this case, the maximum value might correspond to the wrong class and hence, return incorrect vector.

Fig. 3.4 shows an example about how the TF-IDF works in Attack Vector(AV) metric. Suppose *Bluetooth*, *adjacent*, *dongle*, and *lockscreen* are the top four most frequent words exist in the minority values of AV, "Adjacent Network" and "Physical", that build the initial signature word dictionary. After collecting such frequent words, the TF-IDF score of each word corresponding to each potential value is calculated and then normalized. If threshold $th = 0.85$, we exclude the word *dongle* from this dictionary since it does not satisfy the two rules to keep the word in the dictionary as its score corresponding to "Physical" value is 0.2 which is larger than $1 - th = 0.15$. After generating the final dictionary, we look for the signature words in the new text input "*Attackers use Bluetooth to get adjacent access.*", and extract the vectors associated with signature words *Bluetooth* and *adjacent* and take the average to return the TF-IDF vector representation of the text, based on the created dictionary of the signature words.

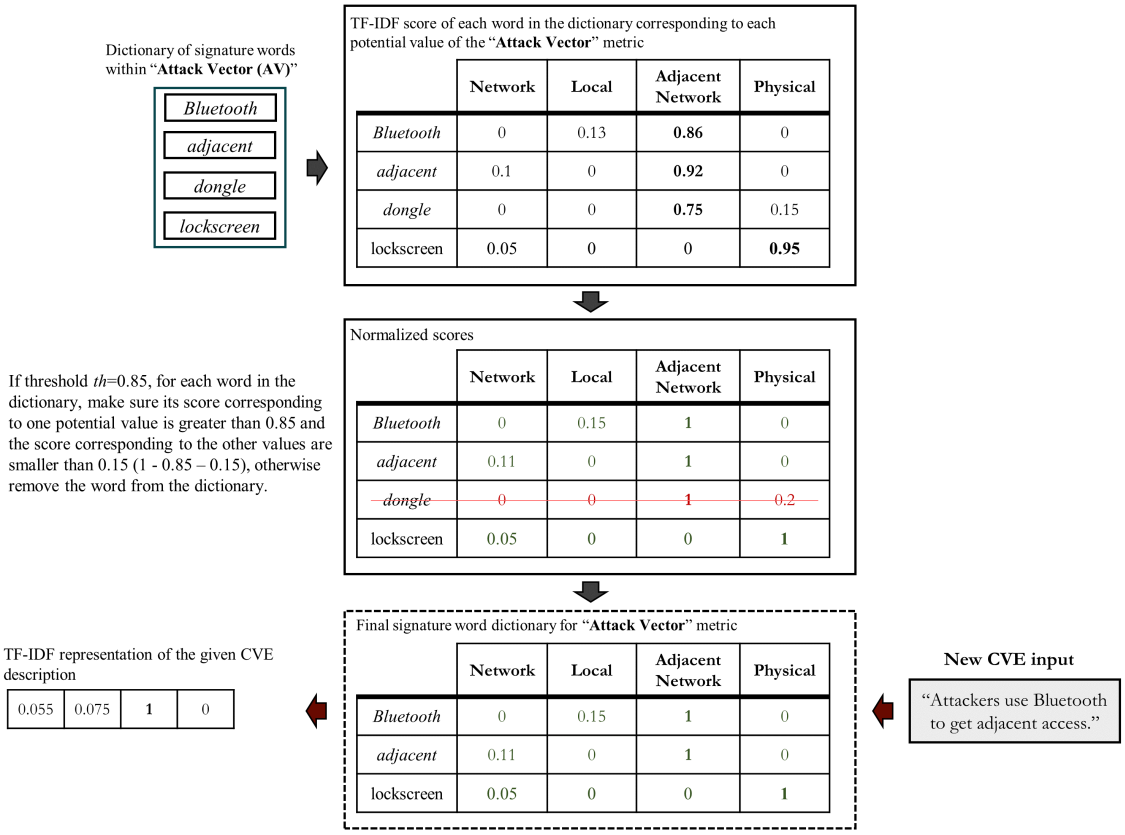


Figure 3.4: An example of TF-IDF module creation and usage.

3.5 Evaluation

In this section, we conduct a comparative evaluation of the performance of our proposed model in predicting the value of each metric separately, based only on CVE human-readable text. We also provide the experimental settings to reproduce the model and discuss the advantages and limitations of the model as well as some suggestions to improve the effectiveness of the CVSS prediction automation.

Experimental Settings

We used NVD dataset that contains 28,090 CVEs reports each assigned by a CVSS vector for training the proposed model. For each report, we create two vector representations, a vector generated by the SecureBERT's tokenizer, and a customized TF-IDF vector utilizing the methodology discussed in the previous section. In addition,

Table 3.3: CVSS metric value prediction results

Model	Metric	AV	AC	PR	UI	S	C	I	A
SecureBERT	Accuracy	**89.57**	**96.04**	**80.59**	**93.01**	**94.60**	83.23	84.35	**87.40**
with	F1-Score (Micro)	**89.45**	**95.96**	**80.58**	**93.01**	**94.39**	**83.23**	84.35	**87.04**
TF-IDF	F1-Score (Macro)	**76.16**	**81.10**	**73.67**	**91.35**	**89.91**	**80.70**	83.58	**72.47**
SecureBERT	Accuracy	89.52	95.85	79.42	91.78	92.06	**83.25**	**84.99**	86.01
without	F1-Score (Micro)	89.38	95.84	79.42	91.78	92.06	83.21	**84.59**	85.91
TF-IDF	F1-Score (Macro)	75.54	80.34	72.03	89.21	86.88	80.37	**83.83**	71.41

we conduct data resampling for minority class(es) within each metric by randomly duplicating samples, to reduce the impact of imbalanced data problem and avoid improper classification while training. We train the model through 10 epochs with a mini-batch size of 12 with a learning rate equal to $1e-5$. The training objective is to minimize the $CrossEntropy$ error using the $Adam$ optimizer.

Model Performance

Table 3.3 shows the comparative performance of our model in predicting every eight values in the CVSS base vector metrics, using a testing dataset of $5,357$ (20% of the main dataset) CVE reports. We evaluated two versions of the model, one with and one without the TF-IDF module. This table shows the accuracy and the F-1 score at both micro and macro level, where micro calculates metrics in each case level by counting the total true positives, false negatives, and false positives, and macro evaluates metrics for each label, and find their unweighted mean. Considering the shortage of key information in CVE reports and imbalance class problem, SecureBERT achieved $80\% - 96\%$ prediction weighted accuracy, $80\% - 95\%$ micro-level, and $73\% - 91\%$ macro-level F1-score.

According to the performance results, the model with the TF-IDF module outperforms the other model indicating the effectiveness of this module in identifying the correct class when the corresponding classes regularly contain specific keywords. Meanwhile, at first glance, the difference between micro and macro metrics reveals the impact of the unbalanced data problem in metrics with a greater incidence of

unbalance. However, the prediction result of some metrics whose micro-macro performance gap is small, yet unbalanced, such as *User Interaction (UI)* and *Scope (S)*, shows the proposed model could successfully identify the correct class. This implies that the CVE reports typically provide the required information for these metrics. On the other hand, the high rate of false positives in both minor and dominating values of some metrics such as *Confidentiality (C)* and *Availability (A)* indicates the lack of key knowledge in CVE reports, leading to a lower rate of correct prediction. Table 3.4 shows the confusion matrix of each eight individual models.

Table 3.4: Confusion matrix for the proposed model

AV		*Predicted* N	L	A	P
True	N	3671	110	36	12
	L	299	961	16	15
	A	36	8	115	0
	L	18	12	4	44

AC		*Predicted* H	L
True	L	4954	99
	H	113	191

PR		*Predicted* L	N	H
True	L	2876	275	100
	N	404	1086	162
	H	58	84	312

UI		*Predicted* R	N
True	R	3667	145
	N	229	1316

S		*Predicted* U	C
True	U	4323	133
	C	169	732

C		*Predicted* L	N	H
True	L	2759	139	201
	N	198	797	55
	H	259	46	903

I		*Predicted* L	N	H
True	L	2346	73	221
	N	154	724	58
	H	296	36	1449

A		*Predicted* L	N	H
True	L	2703	261	43
	N	290	1909	12
	H	56	32	51

Shortages and Limitations

As mentioned earlier, CVEs typically fail to provide all key information about the CVSS metrics. For example, Table 3.5 shows the description of CVE-2012-1516 that is describing a vulnerability in VMware. The "Attack Complexity" metric describes "*the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Such conditions may require the collection of more information about the*

target, the presence of certain system configuration settings, or computational exceptions" [2]. This is a pretty deep and conceptual definition requiring some background knowledge and fine-grain information about the vulnerability to specify if the value of this metric (Attack Complexity) for the given CVE is low or high. According to the vendor report, the basic requirement for this attack to exploit is 4GB of memory, and those with less than 4GB of memory are not affected, hence the "Attack Complexity" for this CVE is rated as "Low". The CVE description (see Table 3.5) failed to provide such detailed information (i.e., the 4GB RAM need), making it pretty obscure even for an expert to predict the metric value based on the provided text.

Table 3.5: Shows an example of uninformative CVE description for the purpose of "Attack Complexity" metric prediction.

CVE-2012-1516: *The VMX process in VMware ESXi 3.5 through 4.1 and ESX 3.5 through 4.1 does not properly handle RPC commands, which allows guest OS users to cause a denial of service (memory overwrite and process crash) or possibly execute arbitrary code on the host OS via vectors involving data pointers.*

Attack Complexity	Low
Reason	The only required condition for this attack is for virtual machines to have 4GB of memory. Virtual machines that have less than 4GB of memory are not affected. Such metric prediction is based on expert opinion.

CVE database comprises many similar reports making both manual or automated CVSS perdition hard or even infeasible, solely based on the description. Some works leveraged other standards such as CWEs to enrich the CVE texts for better prediction. However, this is highly problematic and leads to inconsistency since such standards represent the threats with a high-level view and may not provide fine-grain information about the vulnerability's detailed properties. The complementary solution to improve automated CVSS prediction is to use third-party resources such as vendor vulnerability reports where in some cases, they provide low-level information about the vulnerabilities from different angles such as impacts, platforms, specific permissions, techniques, tactics, and procedures. In this case, a powerful text analytic tool

[2] `https://www.first.org/cvss/v3.0/specification-document`

such as SecureBERT can be utilized to process the context and analyze the semantic relationships, and possibly extract particular actions and features to represent each metric, if the information is provided within the input text properly.

3.6 Conclusions and Future Works

In this work, we proposed a framework that combines the SecureBERT cybersecurity domain-specific language model and a TF-IDF module to identify contextual information and extract key statistical features in order to predict the CVSS base vector for any given CVE based only on the text description. We conducted a rigorous and comparative evaluation and discussed the advantages and limitations of the model in detail, and recommended potential solutions to make the automated CVSS prediction robust and effective. Our approach performs well in detecting the values of particular metrics if the CVEs provide any important information about that metric.

CHAPTER 4: Automated Classification of CVEs to CWEs and to Vulnerability Types

This chapter discusses the importance of enriching CVE reports and propose a model on top of the pre-trained SecureBERT to automatically classify CVEs to CWEs and CVE to vulnerability types (VT). CWEs and VTs both referring to weaknesses which play an important role in CoA inference.

4.1 Introduction

A cybersecurity vulnerability is any flaw in an information technology system, internal controls, or system procedures that attackers can exploit. Cyber attackers can gain access to the systems and acquire data through sources of vulnerability, and hence they are critical in the field of computer security. Unpatchable or zero-day vulnerabilities are the major sources of cybersecurity incidents, which can result in considerable economic losses for enterprises. CVE provides an easy and consistent means for vendors, companies, researchers, and other interested parties to communicate cybersecurity information. These reports are often low-level and product-oriented that typically fail to provide a high-level insight about the underlying threats and therefore, they are not suitable for risk management and cyber threat intelligence. Thus, classifying existing vulnerabilities to common weakness enumerations (CWEs) [1] is a key tool for understanding and mitigating the vulnerabilities. CWEs are designed in a hierarchical form, with weaknesses in lower levels inheriting the traits of their parents. Following this tree-based classification scheme is crucial since it reduces the prediction complexity and provides multi-level information about the threat that is useful in establishing a comprehensive understanding of the vulnerability and obtaining CWE-level mitigation strategies. In addition to the hierarchical classification, CWEs sharing similar properties can be grouped by representing particular behavior, called vulnerability types (VT). This aggregated representation of CWEs is beneficial

when the fine-grain CWE classification is not feasible for a CVE, and/or when just a general threat characteristic is required. Furthermore, as each VT represents several similar CWEs and therefore, the number of VTs is much smaller than the number CWEs, the classification model carries less complexity and is easier to deploy. In addition, in the MITRE guideline, VTs are linked to several common MITRE techniques, which is effective in connecting CVEs to techniques and critical security controls in order to identify technique level mitigations and security controls as the potential course of defense actions against the vulnerabilities. Table A.1in the Appendix shows the full mapping of vulnerability types to MITRE ATT&CK techniques.

Classifying CVE reports to CWEs has previously been done manually, leaving dozens of critical and new CVEs unclassified, yet unpatchable. This significantly limits the utility of CVEs and slows down proactive threat mitigation, due to the scarcity of security experts and a growing number of vulnerabilities every day. Using cutting-edge text analytics and NLP technologies is critical for automating the process of CVE text reports since it demands a deep contextual analysis that traditional machine learning methods cannot provide. Pre-trained language models (LM), such as BERT, have lately received a great deal of interest from the research community for complex text analysis tasks. However, since these models are trained on generic English, they might be inefficient for processing domain-specific documents, such as CVEs. Thus, cybersecurity-specific LMs like SecureBERT can be a viable option for developing a robust and effective predictive model in cybersecurity.

In this chapter, we propose an automated by fine-tuning the SecureBERT to classify CVEs to CWEs. We evaluated the performance of our proposed model through several testing methodologies and conducted comparative research with other similar models to illustrate the efficacy and applicability of our approach.

4.2 Problem Definition

Cyber attacks enable malicious actors to break intended security policies by circumventing protective systems or influencing system resources or behavior. As a result, the attack leads to behavior that violates the victim's intended security regulations. Typically, attackers exploit a vulnerability by abusing an existing weakness in a system. CWE is a hierarchically designed dictionary of software weaknesses for the purpose of understanding software flaws, their potential impacts if exploited, and identifying means to detect, fix, and prevent errors. CWE classes are organized hierarchically, with higher level classes providing higher level attack characteristics and lower level classes inheriting the parent classes' features and adding micro granularity details corresponding to the potential threats. Therefore, determining the best path from a root to lower-level nodes allows for the acquisition of fundamental and functional directions for detecting and analyzing the different properties of a vulnerability.

For example, given 'CVE-2004-0366: A *SQL injection vulnerability* in the libpam-pgsql library before to 0.5.2 allows attackers to execute arbitrary SQL statements.', the description captures the attack action (execute arbitrary SQL statements) within a specific object (libpam-pgsql library) and specifies the consequence (SQL injection). While this low-level, product-oriented description depicts SQL injection exploitation, it falls short of clearly defining the characteristics of this malicious behavior, which is highly required to address possible prevention and/or detection measures. The supplementary CWE (CWE-89: SQL Injection) [1] gives high-level and non-product-specific information by addressing three critical questions: (1) why the attack is used: the system does not properly neutralize special elements; (2) how the attack is used: by changing the intended SQL query; and (3) what the probable results are: access or modify application data; and bypass protection mechanism.

The above-mentioned case is a confirmatory example to show how a CWE can paint

[1] `https://cwe.mitre.org/data/definitions/89.html`

a clear picture of the existing holes in the systems and reveals potential factors leading to vulnerability exploitation. Obtaining these factors is closely associated with the paradigm of pinpointing applicable mitigation or detection methods. For example, we can apply an "accept known good" input validation strategy, i.e., using a set of legit inputs that strictly conform to specifications and rejects the rest, to mitigate SQL injection. Besides, we can detect SQL injection by performing an automated static analysis (e.g., bytecode or binary weakness analysis), dynamic analysis (e.g., database or web service scanners), or design review (e.g., formal methods).

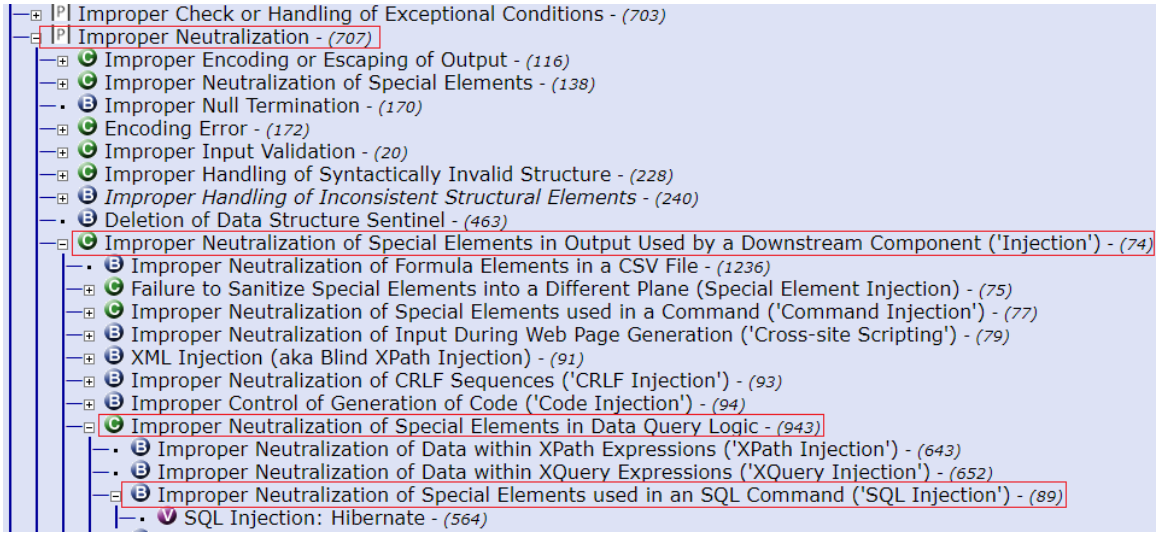


Figure 4.1: It depicts the hierarchical representation of the CWEs. The red boxes show the CWE-89's relatives in the higher levels. This hierarchy plays an important role in understanding the character of the weaknesses in different level of details.

Fig. 4.1 shows the hierarchical representation of the CWEs. Analyzing the path from the root all the way to any node in the lower levels is indispensable since each node reveals different functional directions to learn a weakness. For example, by tracking the path from the root node, CWE-707, to CWE-89, we realize that the SQL injection (CWE-89) is a result of an improper neutralization of special elements in data query logic (CWE-943), where both weaknesses are associated with injection (CWE-74), and the injection itself is the result of improper formation and neutralization of a message within a product before it is read from an upstream component

or sent to a downstream component (CWE-707). Incorporating this thorough knowledge graph helps to maintain countermeasures from different senses, even if the most fine-grain node is not available. For example, assume that only two coarse-grain candidates in different levels of hierarchy, CWE-707, and CWE-74, are available for CVE-2004-0366, while the most fine-grain weakness (CWE-89) is not discovered yet. Although fine-grain SQL injection characteristics is not exposed, investigating the coarse-grain candidates helps to find the common consequences and impacts, and accordingly extract defense actions against improper neutralization and injection (e.g., filtering control-plane syntax from all input). This example explicitly highlights the significance of the existing hierarchical structure of CWEs and shows how useful it is in perceiving the defense actions. A significant number of CVEs are currently mapped to a small set of CWE classes. Currently, about 70% of the CWE classes have fewer than 100 CVEs and about 10% have no CVEs mapped to them, and only 10% have more than 500 CVEs.

4.3 Related Works

A great effort has been made initially by MITRE and NVD[2] to manually classify some CVEs to one or more CWE classes, each one shows critical shortcomings though. Considering the growing number of CVEs and the high labor cost of manual classification, MITRE has classified 2553 CVEs (out of 116K) to 364 CWE classes (out of 719) [1]. On the other hand, NVD has [41] attempted to increase the quantity of CVE classification by mapping about 85,000 CVEs. Although MITRE has classified a smaller number of CVEs compared with NVD, it considers a higher number of CWEs and performs hierarchical and a more fine-grain classification. In the meantime, NVD classified more CVEs but it took a smaller number of CWEs into the account, without addressing the hierarchy.

In the meantime, there have been several research efforts other than MITRE and

[2]National Vulnerability Database

NVD to analyze CVEs to enhance the searching process and to perform CVE categorization. Aghaei *et al.* [27, 28] proposed a hierarchical classification model by utilizing the TF-IDF weights of N-grams extracted from CVE descriptions as the initial weights of a simple feed-forward neural network. This neural network is trained on both MITRE and NVD datasets in a hierarchical fashion and reported the accuracy between 75% and 92%. This work is tested on a limited and selected number of CVEs (10,000 CVEs) and failed to report the model performance at each level. In addition, vulnerability type classification has not been conducted in this work.

Neuhaus *et al.* [42] proposed a semi-automatic method to analyze the CVE descriptions using topic models to find prevalent weaknesses and new trends. The test result reports 28 topics in these entries using Latent Dirichlet Allocation (LDA) and assigned LDA topics to CWEs [42]. This approach shows a highly limited accuracy depending on the CWE type.

Na *et al.* [43] proposed Naïve Bayes classifier to categorize CVE entries into the top ten most frequently used CWEs with the accuracy of 75.5%. However, the accuracy of this limited classification significantly decreases as the number of the considered CWEs increases (i.e., accuracy decreased from 99.8% to 75.5% when the number of CWE classes increased from 2 to 10). In addition, this approach does not consider the hierarchical structure for CWEs, which significantly limits its value. Another classifier was developed to estimate the vulnerabilities in CVEs using the basis of previously identified ones by Rahman *et al.* [44]. This approach uses the Term Frequency-Inverse Document Frequency (TF-IDF) to assign weights to text tokens from the feature vector and Support Vector Machine (SVM) to map CVEs to CWEs. However, they use only six CWE classes and 427 CVE instances. In addition, their classifier does not follow the hierarchical structure for CWEs as well. All these addressed issues and limitations are resolved in this work.

4.4 Challenges

Associating CVEs with CWEs allows cybersecurity researchers to understand the means, assess the impact, and develop solutions to mitigate threats. However, the problem is loaded with challenges. A CVE can be mapped to multiple and interdependent CWEs on the same route, leading to uncertainty. On the other hand, the high-quality mapping information is scarce since CVEs are currently mapped manually to CWEs, which is neither scalable nor reliable. Manual mapping of CVEs is not a practical strategy since new CVEs are introduced at a quick rate. Hence, effective approaches for automating the mapping of CVEs to CWEs are critical for addressing ever-increasing cybersecurity risks.

CVEs are typically short and low-level descriptions written in an advanced language. The format and terminology used in these reports require cybersecurity knowledge and a deep context understanding for automation. Traditional text mining approaches that work based on word frequency fail to adequately capture the context and the semantic relationships between the words. Additionally, the modern NLP tools and pre-trained language models which are trained only on general English corpus with no specific focus on cybersecurity, may not effectively and optimally process such advanced language. Therefore, a domain-specific model that is trained on cybersecurity data can help in more efficient.

Furthermore, the available CVE-to-CWE dataset is highly unbalanced meaning that some CWEs are much more common than others. A portion of this issue is justified, as NVD focuses on more general CWEs that encompass a broader range of specific characteristics, rather than using several distinct CWEs. For example, "CWE-79: Cross-site Scripting (XSS)" is the most frequently reported vulnerability in the NVD classification, accounting for 16,019 CVE reports. Meanwhile, NVD has only used a variant of the XSS vulnerability, the "CWE-87: Improper Neutralization of Alternate XSS Syntax" only one time. NVD's similar use of weaknesses to represent

CVEs demonstrates that several primary CWEs cover specific properties and can thus represent a large group of similar CWEs. However, this does not apply to all CWEs as some of them are not commonly exploited, yet carry unique properties. Therefore, there must be a robust and well-defined approach to be able to classify CVEs to uncommon CWEs appropriately. Table 4.2 shows the distribution of the top 50 most commonly used CWEs by NVD. As depicted, the appearance of a few CWEs such as "CWE-79: Cross-site Scripting (XSS)", "CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer", and "CWE-20: Improper Input Validation" is much higher than other weaknesses.

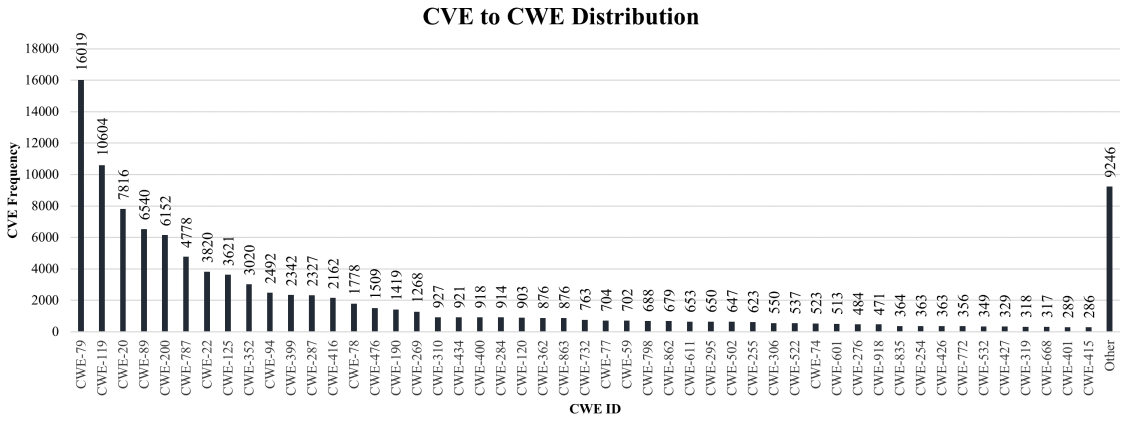


Figure 4.2: the distribution of common CWEs

4.5 Methodology

This section presents two automatic approaches to classify CVEs to CWEs and VTs by fine-tuning the SecureBERT by designing a classification layer on top of SecureBERT.

For CWE classification, the model takes a CVE description as input and returns corresponding CWE classes at each level of the hierarchy. To this end, we developed a top-down strategy, training a classifier on each node (class) at each level in the CWE hierarchical tree. The classifier chooses each node on whether to belong to a different sibling. The primary goal of this hierarchical structure is to direct the

model's attention to the commonalities and contrasts between the sibling nodes. This helps in lowering the model complexity by minimizing the number of output classes during the training procedure, as well as boosting model performance by allowing each classifier to focus on its children without being confused with other siblings' children.

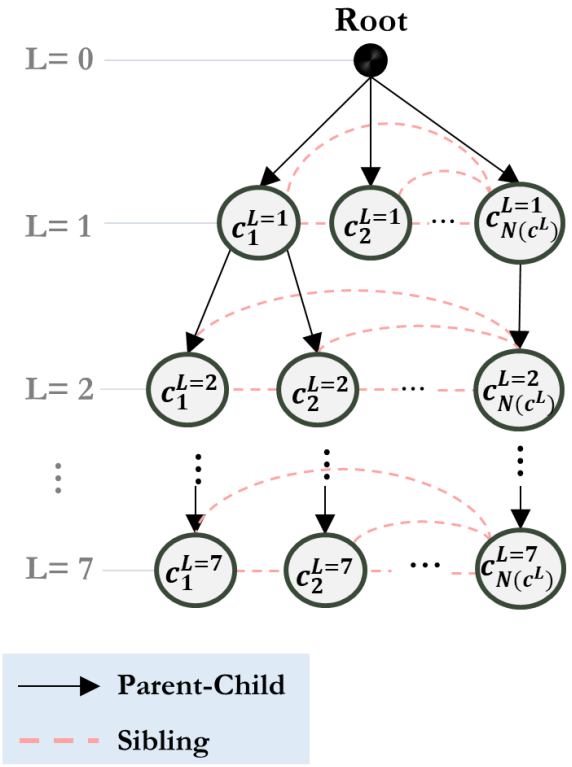


Figure 4.3: the CWE tree structure.

Fig. 4.3 shows the CWE's tree structure in advance and demonstrates the existing levels, and "parent-children" and "sibling" relationships between the nodes. Tracing this tree from left, let's denote the i_{th} CWE at Level L by c_i^L where $i \in \{1, ..., N(c^L)\}$ and $L \in \{1, ..., 7\}$. In our model design, for any given CVE, we aim to start predicting the associated CWEs from the $L = 1$ and trace the CWE hierarchical tree for the correct class at each level to the node in the lowest possible level. Let's denote each CWE as a node c_i^L, and total number of CWEs at level L with $N(c^L)$. Let $G(c_i^L)$ represents all the children of c_i^L:

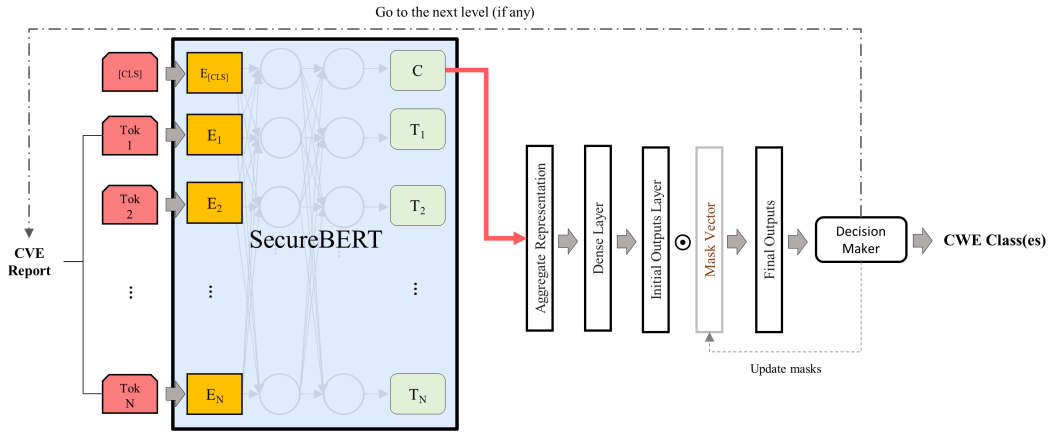


Figure 4.4: CVE to CWE hierarchical classification model design.

The model initially classifies the given CVE to one or more c_i^L and then as a next step, it only focuses on the children of the predicted parent $\mathbf{G}(c_i^L)$ and moves forward until it reaches the most fine-grained level possible (or desired). We use the pre-trained SecureBERT as the base model and add a classification layer on top which trains hierarchically. As depicted in Fig. 4.4, the model takes the CVE description tokenized by SecureBERT's tokenizer and passes it through the transformers stack, and returns the embedding vector of the first token ($[CLS]$ token). This hidden state vector $[CLS]$ is an aggregate representation of the entire input used for classification tasks. This vector is connected to a pooling layer followed by a dense layer of size $N(c^L)$ returning the logits with no activation function, where $N(c^L)$ represents the total number of nodes in layer L.

In order to make the model works in a hierarchical form, we multiply the output logits with a given masking vector MV, and forward the results (final logits) to the decision making component. MV is a sparse vector whose size equals to the initial output layer ($N(c^{(L+1)})$ generated based on the predicted node in the previous level, and $N(c^{(L+1)})$ represents the size of the predicted node's children. Suppose the model classifies an input CVE to node c_i^L. If the total number of nodes (aka, total number of possible CWE classes) in the next level $L+1$ equals to $Nc(^{(L+1)})$, MV would

be a vector of 0s and 1s in which, 1s represent the indices of c_i^L's children nodes in the level $L+1$, and 0s refer to the non-children nodes. The initial value of MV's elements in level $L=1$ would be all 1s, since in the first round of training, there are only parents defined. We multiply this vector to the initial output layer using element-wise multiplication to help the model recognizing which nodes it should be focusing on during training and backpropagation. In other word, this multiplications turns off the nodes which are not the children of previously predicted class(es), hence reducing the complexity by lowering the number of potential classes for each input.

In addition to the hierarchical classification, we design another classification model to classify CVEs to VTs. Following the MITRE guideline[3], similar vulnerabilities often also have the same attack steps and accordingly similar properties in which, the similarity refers to the CVEs associated with similar CWE types. MITRE guideline defines 27 distinct vulnerability types without referring to the CWEs associated with each type. As mentioned earlier, CWEs have two sorts of relationship including parent-children and sibling. We leverage such relationships and manually map CWEs to the corresponding types. Table 4.1 shows our manual effort in mapping CWE to the vulnerability types (VTs). Note that, VTs are mainly defined for the purpose of finding CVEs with common set of techniques used to exploit that is useful in mapping CVEs to MITRE techniques. Therefore, such vulnerability types are not comprehensive and do not cover all existing CWEs.

Fig. 4.5 shows the CVE-to-VT classification model architecture. It follows a similar structure as the hierarchical model with two main differences. First, it is a flat model, and therefore, there is no masking vector. In addition, in the decision-making component, it utilizes Softmax to return the probability of the predicted output. In short, the output size of this multiclass classification model is the fixed number 27 which equals the total number of VTs defined in Table 4.1.

[3] `https://github.com/center-for-threat-informed-defense/attack_to_cve/blob/master/methodology.md`

Table 4.1: Mapping CWEs to vulnerability types.

ID	**Vulnerability Type**	**CWE ID(s)**
1	General Improper Access Control	284, 285, 287, 862, 863
2	Improper Restriction of Excessive Authentication Attempts	306, 307
3	Authentication Bypass by Capture-replay	294
4	Overly Restrictive Account Lockout Mechanism	645
5	Use of Password Hash Instead of Password for Authentication	836
6	General Credential Management Errors	255, 256, 257, 260, 261
7	Cleartext Transmission of Sensitive Information	319
8	Hard-coded Credentials	798
9	Weak Password/Hashing	328, 916
10	General Cryptographic Issues	310, 324, 325, 326
11	XML External Entity (XXE)	611, 776
12	XML Entity Expansion (XEE)	776
13	URL Redirection to Untrusted Site ('Open Redirect')	601
14	Cross-site Scripting (XSS)	79, 692
15	OS Command Injection	78
16	SQL Injection	89, 564, 943
17	Code Injection	94
18	Directory Traversal (Relative and Absolute)	20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40
19	Symlink Attacks	59, 61, 62, 64, 65, 73, 363
20	Untrusted/Uncontrolled/Unquoted Search Path	426, 427, 428
21	Unrestricted File Upload	434, 351, 436, 430, 73, 183, 184
22	Deserialization of Untrusted Data	502
23	Infinite Loop	835
24	Cross-site Request Forgery (CSRF)	352
25	Session Fixation	384
26	Uncontrolled Resource Consumption	400, 664, 770, 771, 779, 920, 1235, 410
27	Server-Side Request Forgery (SSRF)	918

4.6 Evaluation

In this section, we undertake a comparative evaluation of our proposed model's performance in classifying CVEs to CWEs using only human-readable language from CVEs. We train two different classification models, a hierarchical CVE to CWE classification model and a CVE to vulnerability types classification model, and perform a comparative evaluation on each. Additionally, we provide experimental settings for reproducing the model and discuss the model's strengths and drawbacks.

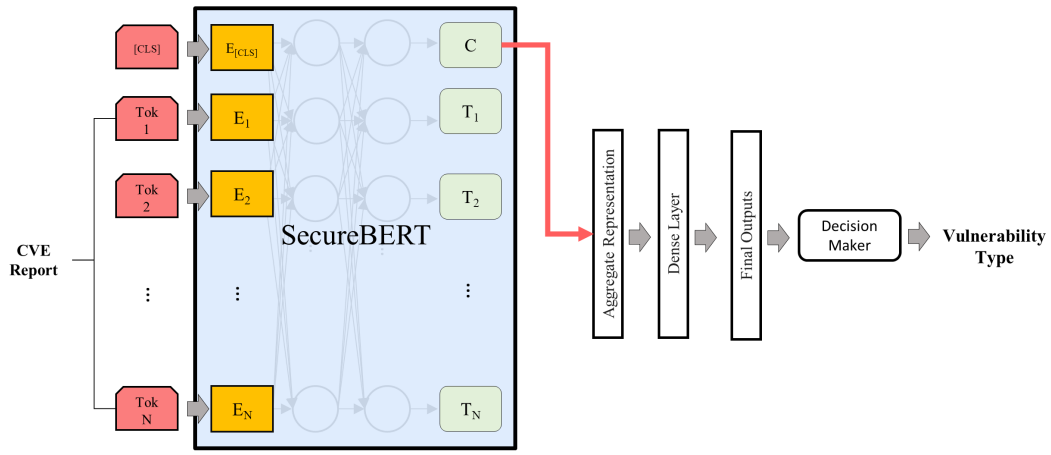


Figure 4.5: CVE to vulnerability type classification model design.

4.6.1 Hierarchical CVE to CWE Classification Model

We use a list of 63,481 CVE descriptions to train the hierarchical model and classify them to one or more CWEs at each level of the hierarchy. Since the lowest level, CWE used by NVD is $L = 5$, we train five models separately corresponding to each level. For each model, we use Binary Cross Entropy (BCE) with logits as loss function for this multiclass multi-label classification task. This loss function takes the raw logits of the model (without any non-linearity) and applies the sigmoid internally. We use 10 epochs with Adam optimizer and the initial learning rate of $3e-5$. For training each model at level $L+1$, we further fine-tune the trained model in level L instead of the raw pre-trained SecureBERT, to ensure it carries the most recent information about the CVE texts. In the decision-making component in the output layer of the model, we perform min-max normalization to produce a probability score corresponding to each output unit. We have conducted a two-round evaluation to show how well our model performs when hierarchically classifying CVEs to CWEs. In the first round, we evaluate the classification using standard metrics to monitor how the model performs in the top one prediction at each level of the hierarchy. In the second round, we monitor the performance of the model not in the first returned output, but in the top K outputs.

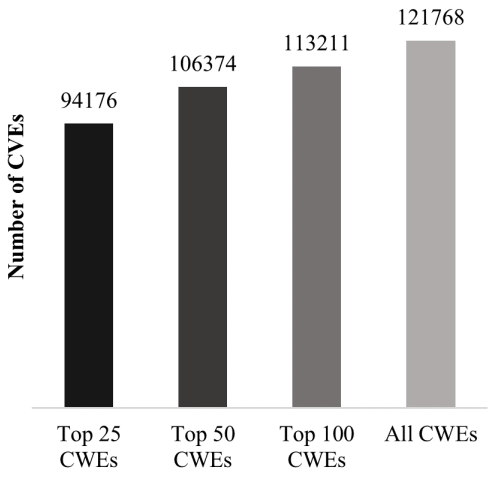


Figure 4.6: Distribution of CWEs in CVE to CWE classification.

NVD dataset had classified 121,768 CVEs to 296 CWEs as of the date of authoring this article. However, there is a significant disparity in the frequency with which CWEs are used, indicating that some CWEs are significantly more prevalent than others. This implies that infrequent CWEs are either rare or have characteristics in common with one of the frequent CWEs. As depicted in Table 4.6, 77.3% ((94,176 CVEs)), 87.4% (106,374 CVEs), and 93% (113,211 CVEs) of the CVEs have been classified to top 25, 50, and 100 CWEs, respectively. Note that, in calculating true predictions, if the instance was originally labeled by more than one CWE (multilabel instances), and the model catches one with no false positives, it is considered correct classification as any correct prediction can lead to the correct label in the lower levels.

In the hierarchical model, we classify CVEs to the common CWEs and evaluated the model in different stages. Table 4.3 shows the performance of the hierarchical model in classifying CVEs to the top 25, 50, and 100 most frequent CWEs. According to this table, the success rate of our proposed model ranges from 90.77% to 97.51% in terms of HR@1 (accuracy), and 84.91% to 96.08% in terms of F1@1, concerning the CWE space and the level of hierarchy. Such results demonstrate a higher success rate in classifying CVEs belonging to the top 25 CWEs due to the higher number of

samples available for training the model. The hierarchical model provides researchers

Table 4.2: SecureBERT performance evaluation on hierarchical classification of CVEs to top 25, 50, and 100 CWEs where HR indicates the hit rate (accuracy in this case) and F1 refers to the F1-score.

	Top 25 CWEs			**Top 50 CWEs**			**Top 100 CWEs**		
L	No. CWEs	**HR**	**F1**	No. CWEs	**HR**	**F1**	No. CWEs	**HR**	**F1**
1	8	91.80	91.11	8	91.24	91.15	9	90.77	90.31
2	18	94.55	91.38	30	93.57	87.48	43	91.11	85.89
3	14	97.51	96.08	36	95.46	85.89	63	94.24	84.91
4	6	95.22	94.35	13	94.02	85.21	27	94.57	90.81
5	-	-	-	3	91.53	91.53	7	90.80	90.96

with a variety of classification granularity levels. In other words, researchers may need different levels of information to characterize the properties of CVEs. As a result, offering level-specific knowledge about CVEs can provide a multitude of CVE characteristics. CVEs, on the other hand, may exploit multiple weaknesses, which cannot be addressed if they are assigned to a single CWE. As a result, hierarchical models are highly useful in identifying such weaknesses and providing broader insight into exploit to help in better assessment and, consequently, defense planning. As previously noted, CVEs may be linked with multiple CWEs, however NVD has only assigned them one. The similarity between CVE characteristics that are mapped to different CWEs and possibly the similarity across CWEs leads to different classification outcomes that are not necessarily incorrect. Since there is no ground truth for the additional corresponding CWEs with CVEs, we have conducted another experiment trying to show the performance of the model in classifying CVE to the correct class(es) in the "top K" model prediction outputs. The correct class(es) refers to the labels originally provided by NVD. For this experiment, we used standard information retrieval metrics including hit rate at K (HR@K), precision at K (P@K), and recall at K (R@K).

Table 4.3 shows the performance of the proposed hierarchical model in classifying 25,915 CVEs to top 25, 50, and 100 CWEs in different levels of hierarchy. Each table shows the HR@K, P@K, and R@K in each level of hierarchy, as well as the number

Table 4.3: SecureBERT performance evaluation on hierarchical classification of CVEs to top 25, 50, and 100 CWEs.

Top 25 CWEs							
L	No. CWEs	**HR@2**	**P@2**	**R@2**	**HR@3**	**P@3**	**R@3**
1	8	96.82	94.44	92.44	98.92	95.83	94.29
2	18	94.95	94.13	90.28	95.11	94.54	90.79
3	14	97.51	96.63	95.69	97.51	96.69	95.63
4	6	95.42	95.63	93.18	95.61	95.12	95.06

Top 50 CWEs							
L	No. CWEs	**HR@2**	**P@2**	**R@2**	**HR@3**	**P@3**	**R@3**
1	8	95.91	94.41	92.77	98.18	96.46	93.72
2	30	94.29	89.84	86.78	94.33	91.55	87.58
3	36	95.26	86.25	85.95	95.26	91.31	88.60
4	13	94.73	90.44	88.62	94.73	92.02	86.27
5	3	91.53	91.09	89.24	1.0	1.0	1.0

Top 100 CWEs							
L	No. CWEs	**HR@2**	**P@2**	**R@2**	**HR@3**	**P@3**	**R@3**
1	9	95.12	94.21	93.38	97.51	95.17	91.20
2	43	92.30	87.11	87.59	92.30	89.55	88.59
3	63	94.61	87.73	85.55	94.61	88.31	86.16
4	27	95.44	90.81	91.74	95.44	88.96	91.74
5	7	91.11	88.93	86.71	91.11	89.49	88.19

of CWE classes exist at each level. In classifying CVEs to top 25 CWEs, which covers the majority of CVEs, our model successfully classified CVEs to the correct CWEs with the hit ratio at top K (HR@K) ranged from 94.95% up to 98.92%, P@K between 94.13% and 96.69%, and R@K between 90.28% and 95.69%, with K$\in \{2, 3\}$ in different levels. Despite the promising performance in classifying CVEs to top 50 and 100 CWEs, lower performance (particularly in F1-scores) compared to top 25 can be justified by two reasons. First, the additional CWE classes have much lower number of CVEs in the training set causing less accurate prediction, Furthermore, many of such additional CWEs share common characteristics with the frequent CWEs and as a result, CVEs are mostly mapped to the common ones by NVD. Therefore, as we use such mappings as labels in our training set, our model tends to classify CVEs that originally associated with uncommon CWEs to frequent CWEs instead.

4.6.2 CVE to Vulnerability Type (VT) Classification Model

Similar to individual CWEs, VTs are not distributed uniformly in NVD dataset. As depicted in Fig. 4.7, about 72.3% (41,240) and 88.03% (50,174) of the CVEs are associated with top 5 and top 10 VTs, respectively. In addition, four VT IDs including

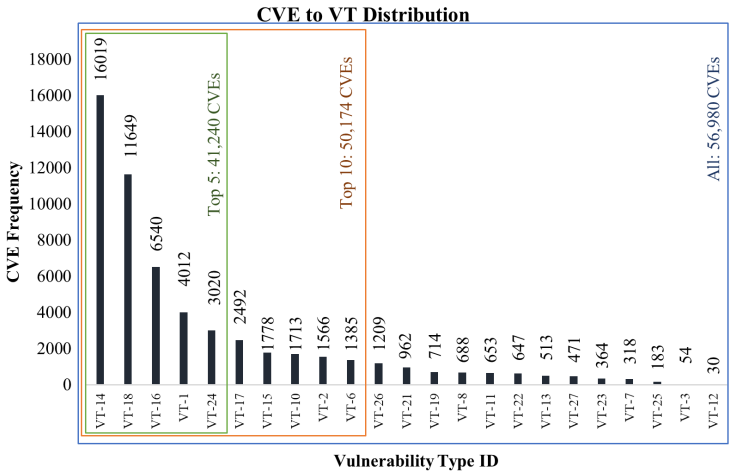


Figure 4.7: Distribution of CWEs in CVE to CWE classification.

4, 5, 9, and 20 do not have any corresponding CVE in the NVD dataset. For training the CVE to VTs, we use 70% of the available CVE descriptions to classify them to one of the top 5, 10, and all 23 VTs provided in Table 4.1.

Table 4.4: SecureBERT performance evaluation in classifying CVEs to top 5, 10, and all vulnerability types.

Top VT	**HR@1**	**P@1**	**R@1**	**HR@2**	**P@2**	**R@2**
Top 5	93.12	91.34	89.08	95.21	92.31	89.46
Top 10	91.66	88.63	86.79	93.71	90.03	87.11
All	87.53	85.79	84.56	90.35	87.32	86.80

Similar to the previous model, Table 4.4 shows the performance of CVE-to-VT model in terms of HR@K, P@K, and R@K.

4.7 Conclusions and Future Works

In this work, we designed two models on top of the SecureBERT to hierarchically classify CVEs to CWEs and to classify CVEs to the groups of CWEs, aka vulnerability types (VT), that share similar properties. We conducted a thorough and comparative evaluation in different stages and discussed the advantages and limitations of each model in detail. In short, we made the following contributions:

1. Design a hierarchical classification model on top of SecureBERT to classify

CVEs to different CWEs in each level of CWE's hierarchical design.

2. Manually categorizes similar CWE to different groups known as vulnerability types where each category represents a particular set of behaviors and properties.

3. Build a classification model on top of SecureBERT to classify CVEs to vulnerability types

4. Conduct a comparative and thorough evaluation of both models and provide the advantages and limitations in detail.

In the future, we plan to leverage the third party reports in order to further enrich the descriptions for classifying CVEs to CWE and VTs.

CHAPTER 5: Automated Context-based Classification of CVEs to Functionalities

In Chapter 4, we proposed a model for partially enriching vulnerabilities by classifying them to CWEs and vulnerability types. This exemplifies the CVEs properties in terms of the underlying weaknesses that an attacker may exploit and provides granular information regarding the potential weakness-level impacts should it be exploited.

In this chapter, we aim to further improve the enrichment by building a model that identifies the attacker behavior in order to map the CVEs to the common functions that an attacker could try to access, based on the known functionalities defined by the MITRE guideline. Therefore, we introduce a semi-automated approach to collect and annotate data and then utilize the generated dataset to build a predictive model on top of the SecureBERT to classify CVEs to functionalities.

5.1 Introduction

Enriching the CVEs at CWE-level alone do not provide sufficient information about the technique, tactic, and procedures, and also omits crucial details about the common functions that an attacker may be attempting to gain access. Notably, vulnerability types can help in connecting CVEs to particular MITRE ATTCK techniques which are related to pre-attack, by adhering to the MITRE guidelines. According to the guideline, however, several common vulnerability types including "General Improper Access Control", "Directory Traversal (Relative and Absolute)", "Symlink Attacks", and "Cross-site Request Forgery (CSRF)" are still required to connect to functionality to improve CVE enrichment, to deliver the corresponding MITRE technique, and infer the corresponding course of actions. In the meantime, post-attack knowledge is also essential for characterizing the CVEs. For a vulnerability to be exploitable, it must grant the attacker a previously unavailable functionality. Identifying such cybersecurity functionalities are important since attackers who exploit that particular vulnerability or similar ones typically seek access to the same functionality in every

Table 5.1: List of common functionalities defined by MITRE.

Common Functionalities		
Modify Configuration	Create Account	Disable Protections
Restart/Reboot	Install App	Read from Memory
Obtain Sensitive Information: Credentials		Password Reset
Obtain Sensitive Information: Other Data		Read Files
Delete Files	Create/Upload File	Write to Existing File
Change Ownership or Permissions		
Memory Modification (Memory Buffer Errors, Pointer Issues, Type Errors, etc.)		
Memory Read (Memory Buffer Errors, Pointer Issues, Type Errors, etc.)		

exploits.

The MITRE guideline introduces sixteen functionalities, each of which describes a specific action that the attacker can undertake if the vulnerability is exploited, such as disabling protection mechanisms, installing apps, and obtaining sensitive information, as depicted in Table 5.1. In the meantime, functionalities defined by MITRE are not completely independent and they may share mutual characteristics (dependencies). For example in cybersecurity language, "reading a file" by an attacker can infer "obtaining sensitive information". CVE descriptions typically deliver such capabilities either implicitly or explicitly. For example, "*attacker uninstalled antivirus software*" and "*attacker compromise the firewall's functionality*" both refer to manipulating the protection systems to work inefficiently. Therefore, studying CVE descriptions would suffice to gather such information. These functionalities, if recognized by the defensive side, contribute to improved vulnerability management. In addition, similar to vulnerability types, functionalities are also linked with a range of MITRE techniques (see Table B.1 in Appendix). This allows cybersecurity researchers to better articulate the exploitation process of a vulnerability, resulting in more effective cyber threat intelligence and defense planning. Section 5.5.1 will go over more details about the functionalities.

When it comes to natural language processing, there are numerous ways for extracting information from unstructured text such as part of speech tagging (POS),

named entity recognition (NER), and dependency parsing (DP). However, capturing the particular form of text sequences (e.g., sentences or statements) using stand-alone methods such as POS, NER, and DP requires well-defined rules. The critical aspect to remember when extracting text to represent functionalities is to preserve both the semantic relationship and the context. Therefore, rule-based information extraction solely based on the above-mentioned methods cannot be effectively applied to this problem.

Semantic role labeling (SRL) [45], is a semantic parsing task aimed at identifying the predicate-argument structure of each predicate in a phrase. For example, it predicts important relationships between predicate such as *who* did *what* to *whom*, *where* and *when*, and so on. SRL, in particular, aims to recognize arguments and classify their semantic functions in the presence of a predicate leveraging POS and NER. SRL is a useful method for obtaining semantic information that can be applied to a variety of natural language processing (NLP) tasks, such as neural machine translation, question answering, discourse relation sense classification, and text relation extraction. Thus, SRL is an effective approach to extract information from the CVE description for the purpose of generating a representative labeled dataset for training a model than can classify cybersecurity text to its functionalities.

In this chapter, we present a novel approach for identifying context-dependent threat actions within a CVE text and mapping it to functionalities. Therefore, we begin by introducing a data collection, labeling, and annotation framework by utilizing an off-the-shelf SRL tool. Then, on top of the SecureBERT, we develop a novel classification model that takes the generated threat actions and classifies them to the sixteen functionalities defined by MITRE guidelines, according to the given context. In the end, we conduct a multi-step performance evaluation and discuss the advantages and limitations of this classification model.

5.2 Problem Definition

As previously stated, our goal is to create a model that will automatically classify CVEs to functionalities. To that end, because no dataset exists for training such a model, we first define linguistic structure and create a dataset using SRL, and then employ that dataset to train and evaluate the predictive model.

Functionalities are defined as a set of malicious actions that attackers perform when exploit a vulnerability, where each action can be represented by different syntax and terminologies. In the meantime, an action may imply different meanings within different contexts. For example, as depicted in Fig. 5.1, "deleting a file" in "*<attacker deletes a file>*" implies a malicious action, in "*<web admin deletes a file>*", it is a benign action, and in "*<attacker tricks the web admin to delete a file>*" addresses a malicious action again. In another example, consider the *"read files"* as an action and its meaning in two different contexts such as *"Attacker abuse logprop?file = /.. to read files"* and *"Attacker tricks the victim to read files"*. In the former context, reading a file is exactly the main objective of the attacker (threat action) referring to "reading data from the file", while in the latter sentence, reading file is done by the victim and tricking the victim to achieve other malicious goal such as phishing or executing arbitrary code is the main threat action. In order to understand the

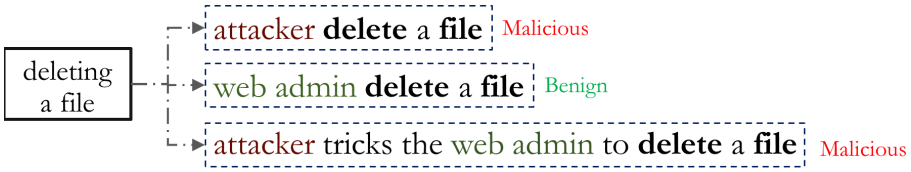


Figure 5.1: Shows the different implication of "deleting a file" action in different contexts.

actual implication and semantic meaning of an action, it is crucial to always analyze a text within the surrounding context. Similarly for classifying actions within a CVE description to functionalities, it is required to consider threat actions within entire description to precisely learn the corresponding functionality.

Semantic Role Labeling (SRL) is a powerful NLP tool to extract such actions from an unstructured text. Off-the-self SRL models such as AllenNLP SRL [1] are able to break down a text into sentences according to the verbs, and then captures the words or phrases as arguments, and return their semantic role in that particular sentence. For example, pre-trained SRL divides "*attacker tricks the web admin into deleting a file*" into two sentence based on two existing verbs, *tricks* and *delete*, as "*attacker* ***tricks*** *the web admin to delete a file*" and "*the web admin* ***delete*** *a file*". As depicted in Fig. 5.2, in the first sentence, it identifies *attacker* as the subject (ARG0), *tricks* as the verb (V), *the web admin* as the object (ARG1), and *to delete a file* as the following attribute (ARG2) of the object. Similarly in the second sentence, SRL identifies *the web admin* as the subject, *delete* as the verb, and *a file* as the object, It worth noting that, there are multiple arguments (ARGs) returned by the SRL

Figure 5.2: SRL breaks down a text into the words or phrases as arguments and return their semantic role in the sentence.

based on PropBank [46], which is a corpus of text annotated with information about basic semantic propositions, as demonstrated in Table 5.3. However, the implications of all arguments is beyond the scope of this study and can be find in PropBank documentations.

This capability makes SRL an effective framework for extracting threat actions from text to generate a labeled dataset. Suppose we aim at retrieving documents that are referring to "*delete file*". Traditional bag-of-words techniques may return documents with large noise such as "*admin* ***delete*** *access to the* ***file***". Similarly, simple word

[1] `https://demo.allennlp.org/semantic-role-labeling`

Table 5.3: PropBank proposition definitions.

Arguments	**Meaning**
V	verb
ARG0	giver (subject)
ARG1	thing given (object)
ARG2	entity given to
ARGM-TMP	when?
ARGM-LOC	where?
ARGM-DIR	where to/from?
ARGM-MNR	how?
ARGM-PRP/CAU	why?

matching is highly limited that fails to detect the phrase if minor difference exist, such as "*delete a file*" or *delete .exe file*. However, instead of creating a large number of rules considering all possible text variations (which is potentially impractical), we may define few rules and instructions based on the target corpora to efficiently employ SRL to extract such information. In other words, for each functionality, we define a set of representative words or phrases and then employ SRL to extract sentences corresponding to each functionality. To maximize the accuracy and simplify the extraction, we only define representative verbs (V), subjects (ARG0), and objects (ARG1) and utilize SRL to return sequences of sentences. For example, to represent malicious action "*deleting a file*", we extract sentences with the following elements:

Verbs: delete, remove, erase

Subjects: attacker, remote attacker, unauthenticated user

Object: file, folder, directory

Accordingly, we retrieve the sentences that its verb is one of the defined verbs and its ARG0 and ARG1 contains any of the given subjects and objects respectively. Table. 5.4 shows the a few examples of retrieved sentences:

In typical text classification tasks, natural language processing (NLP) models automatically capture the semantic relationships between tokens during training with a standard dataset. Based on the input text, the model could classify a CVE into

Table 5.4: Examples of retrieving sentences based on the given rules using SRL.

Deleting a file
'*remote attackers delete log files*'
'*malicious user removes the directory*'
'*unauthenticated user clears registry folder*'

one or more classes in this context. In the absence of a standard dataset, we must first collect sufficient data to construct the dataset and then train a model to automate the process. As previously stated, the dataset must include both malicious action (e.g., *delete a file*) and the surrounding context (e.g., *attacker tricks the web admin into deleting a file*) to successfully enhance the semantic understanding and capture alternative "action representations". For example, an effective NLP model must recognize the semantic relationship between "*attacker disabled protection mechanism*" and "*attacker compromise the firewall's functionality*". Thus, to develop such a robust model, it must be trained on representative, consistent and accurate labeled data. A standard dataset for this task should consist of labeled text records, that each represents a specific functionality. To create such a dataset, we first conduct a statistical text analysis to identify the common terminologies used in CVE descriptions in terms of subject (malicious actor), verb (action), and object (cyber object). This helps to identify the corpus domain and establishing the target dataset's initial linguistic structure. Leveraging this statistical knowledge, we develop a framework utilizing SRL and an off-the-shelf tool called EXTRACTOR (see Section 5.5.2).

We manually go through the observed common terminologies to define rules and extract different statements to represent each functionality. In this context, we utilize EXTRACTOR to extract short statements (so-called content) from longer documents (so-called context) given the discovered terminologies. The purpose of this procedure is to find common statements which imply each functionality in order to train the model to learn the pattern. For example, we extract sentences such as "*remote attackers delete files*" and "*a user without administrator privileges delete arbitrary*

files" to represent "Delete File" functionality. In addition, to further generalize the dataset and improve data quality, we also manually extract additional textual data for each functionality. Further details on data collection process will be provided in Section 5.5).

After data collection, we propose to construct a model that takes two inputs as *content* and *context* and returns the corresponding functionality of the short *content* text within the long *context* document by leveraging SecureBERT and a classification model on its top. The content is defined as a short and precise text addressing a particular action whereas context is usually an longer text in terms of size containing more information, which provides enriched content while also may contain some irrelevant information (noise). A content may have multiple meanings or represent different intentions or concepts depending on the context. A particular action, on the other hand, can also be represented using a variety of terminologies and phrases. For example, *view the file*, *observe the file*, and *open the file* can infer the same meaning as "reading the file" in cybersecurity language when refer to a specific action. Therefore it is critical to identify such meaning and representation by properly understanding the content and capture its semantic relationship with the context. We train our model so that it captures the contextual meaning of the threat actions using two inputs. After training the model and tuning the hyperparameters, this model can take either two inputs, the content and the context, or just one context input. When it receives two inputs, similar to training, it returns the functionality associated with the content "within" the context. For example, the model can use a specific action expressed as a brief text or a sentence from a CVE description as the content and the entire description as the context in order to predict the functionality. On the other hand, the model can also take only one document (e.g., CVE description) as context without any content input and return the potential functionalities associated with the input.

5.3 Related Works

Chen et al. [47] proposed a model to extract threat actions by using information retrieval techniques. To capture threat actions, this study use word vector, tagging, and filtering algorithms. The proposed solution automatically generates a key threat action list as the foundation of the ontology, uses a two-stage key threat action extraction technique, and uses word vector models for key threat extraction. This work labels tokens in a phrase with their grammatical word categories using part-of-speech tagging, but it does not maintain grammatical links between them.

In [48], the authors present a mechanism for automatically extracting threat actions from APT reports and producing TTPs. Threat actions are extracted from APT reports using a BERT-BiLSTM-CRF-based extractor, and these extracted threat actions are then mapped to ontology to construct their related TTPs using TF-IDF. The actions, which include the subject, verb, and object, are extracted using EX-Action. Additionally, it offers a technique for extracting entity relations, which connect entities contextually and semantically. This approach has a problem with its overreliance on semantic and part-of-speech analysis, which can miss some threat actions and fail to identify pronoun referents.

Ayoade *et al.* [49] leveraged natural language processing techniques to extract attacker actions from $18,257$ threat report documents generated by different organizations and automatically classifies them into standardized tactics and techniques. The lack of labeled data and non-standard report formats are the main challenge this paper addresses using the bias correction mechanism approach. In this work, text descriptions of reports are tokenized, and the TF-IDF score for each word is calculated and applied different bias correction mechanisms to overcome non-standard format. Then using the SVM classifier, they classified 78% of the reports correctly to their corresponding techniques and tactics.

5.4 Challenges

The most critical challenge of this work is the lack of labeled data. As of now, the only available data is 840 CVEs [2] mapped to a set of MITRE techniques and tactics by MITRE in which, labels do not precisely correspond to particular functionalities. Therefore, it is required to collect and annotate a standard dataset manually to train the predictive model.

Data collection and annotation, on the other hand, are challenging problems since they require extensive text analysis to fully grasp the language structure of the target domain (CVEs) and to extract relevant text patterns for concise dataset generation.

The MITRE guideline-defined functionalities are not independent, and some of them may share similar characteristics. Thus, a single statement (e.g., a threat action) may have different implications in different contexts and correspond to multiple functionalities, which can lead to confusion and inconsistency during dataset generation and training. To distinguish similar texts, a systematic data collection and a strategic training model design are required.

5.5 Data Assessment and Annotation

As mentioned earlier, there is no off-the-shelf data for training a supervised method to predict the CVEs' corresponding functionality. Therefore, we utilize semantic role labeling along with some manual work to collect, analyze, and annotate data for this prediction task. In this section, we introduce a method to generate labeled dataset leveraging semantic role labeling along with expert knowledge for the purpose of classifying CVEs to functionalities. In this context, we collect the corresponding subjects, verbs, and objects (SVOs) for each functionality from CVE description. Then, leveraging a threat action extractor tool called EXTRACTOR [50], which is built on top of the BERT for semantic role labeling in cybersecurity, we extract text

[2] https://github.com/center-for-threat-informed-defense/attack_to_cve/blob/master/Att&ckToCveMappings.csv

containing such SVOs associated with each functionality.

5.5.1 Functionality Documentation

MITRE guideline has provided sixteen functionality names [3] without any We define sixteen different most common functions an attacker may be trying to gain access to through the exploitation known as functionalities. If f_z denotes the functionality f with index z, let's define the functionalities as follows :

1) Create Account ($f_{z=1}$): the act of unauthorized creation of new accounts or adding new users to the victim system done by attacker.

2) Create Or Upload File ($f_{z=2}$): the act of unauthorized creation or uploading any file to any system for any purpose done by attacker.

3) Delete Files ($f_{z=3}$) : the act of unauthorized deletion or destruction of any information including but not limited to files, contents, data, etc., done by attacker. It is different from data manipulation.

4) Disable Protections ($f_{z=4}$): the act of causing any malfunction, interruption, or abnormality in any security/defensive process or system such as anti-viruses, anti-malware, authentication procedures, firewall, security checks, etc., done by attacker.

5) Install App ($f_{z=5}$): the act of delivering and/or installing any malicious application or configuration on victims system causing further threats, done by attacker.

6) Memory Modification (Memory Buffer Errors, Pointer Issues, Type

[3] `https://github.com/center-for-threat-informed-defense/attack_to_cve/blob/master/methodology.md`

Errors, etc.) ($f_{z=6}$): the act of any invalid modification, manipulation, and/or write to the memory (e.g., buffer, kernel, memory locations, pointers, etc.) leading to memory issues such as buffer-over read, memory crash, buffer overflow, etc., done by attacker.

7) Password Reset ($f_{z=7}$): the act of manipulating account such as modifying credentials (e.g., ID, username, password, email account name, etc.) for any purpose, done by attacker.

8) Change Ownership or Permissions ($f_{z=8}$): the act of changing file ownership, and/or modifying access permission (access controls) for any purpose, done by attacker.

9) Modify Configuration ($f_{z=9}$): the act of modifying, editing, and manipulating any systems configuration and/or settings causing further threats, done by attacker.

NOTE 1: The actions and intention in "Install App" and "Modify Configuration" are quiet similar sharing similar techniques (not same), and since there is no ground truth available to distinguish them, we combine these two and considered them as a single functionality.

10) Obtain Sensitive Information - Other Data ($f_{z=10}$): the act of obtaining any non-credential sensitive information without authorization via any method (unauthorized access to files, databases, memory, etc.) for any purpose, done by attacker.

11) Obtain Sensitive Information - Credentials ($f_{z=11}$): the act of obtaining any user/system credentials without authorization via any method (unauthorized access to files, databases, memory, etc.) for any purpose, done by attacker.

12) Read From Memory ($f_{z=12}$): the act of unauthorized reading any information or data from memory for any purpose, done by attacker.

NOTE 2: "Obtain Sensitive Information - Other Data" and "Read From Memory" share exactly the same MITRE technique. However, since the action and the purpose might differ, they are considered as separate functionalities.

13) Read Files ($f_{z=13}$): the act of unauthorized reading any information including from files, done by attacker.

NOTE 4: This functionality and "Obtain Sensitive Information" (($f_{z=10}$) and ($f_{z=11}$))" are mainly using "Read" action mentioned in the previous functionalities, hence they share mutual characteristics with each other in terms of common MITRE techniques.

14) Memory Read (Memory Buffer Errors, Pointer Issues, Type Errors, etc.) ($f_{z=14}$): the act of any invalid reading from the memory (e.g., buffer, kernel, memory locations, pointers, etc.) leading to memory issues such as buffer-over read, memory crash, buffer overflow, etc., done by attacker.

15) Restart Or Reboot ($f_{z=15}$): the act of crashing, shutting down, rebooting any system often leading to denial of services, done by attacker.

16) Write To Existing File ($f_{z=16}$): the act of modifying the content of the

existing file for any purpose, done by attacker.

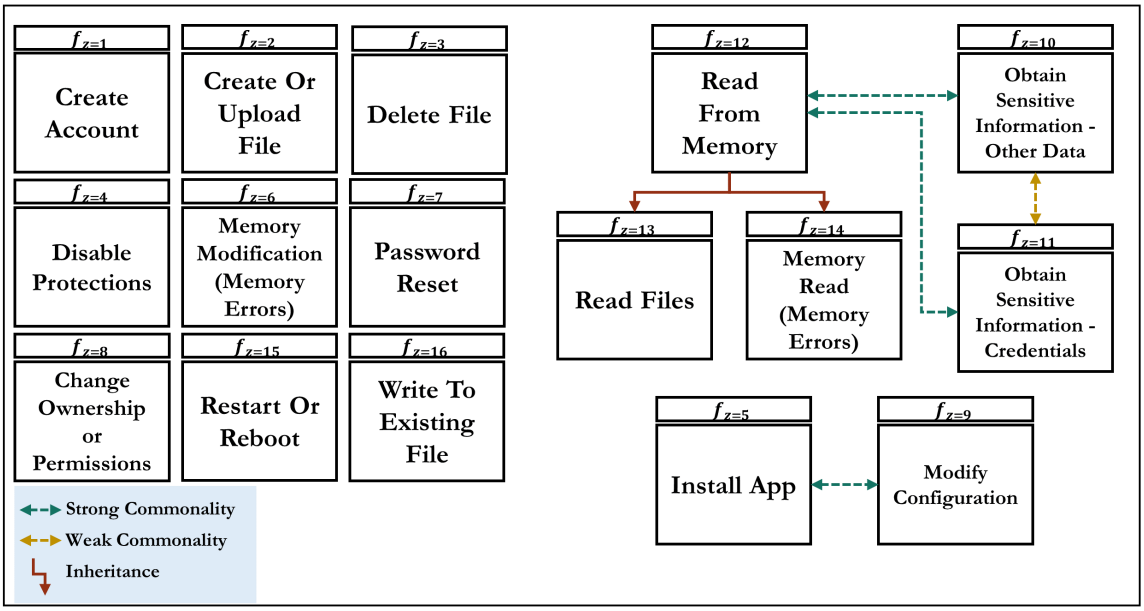


Figure 5.3: Shows the functionalities and their relationships. In relationship definitions, The term "inheritance" denotes that the child functionality inherits all of its parent's characteristics in addition to its own unique ones. The characteristics refers to the same threat action(s) and/or same MITRE technique(s). "commonality" on the other hands refers to semantic similarity between two functionalities, but not necessarily the same behavior.

Fig. 5.3 demonstrates all of the functionalities and their associated dependencies. We have defined two dependencies such as commonality and inheritance. The term "inheritance" denotes that the child's functionality inherits all of its parent's characteristics in addition to its own unique ones. The characteristics refer to the same threat action(s) and/or the same MITRE technique(s) the functionalities are associated with. For example, to conduct *out-of-bound-read*, which is associated with functionality "$f_{z=14}$: Memory Read (Memory Errors)", an attacker must conduct the *read* action which is also associated with functionality "$f_{z=12}$: Read From Memory", such as *reading arbitrary memory* or *reading kernel memory*. On the other hand, commonality refers to the semantic similarity of two classes which means both classes can describe similar impact or action, but not necessarily the same. If the commonality is strong, it implies that the impact can be the same and in some cases cannot

be differentiated. For example, an attacker can install an extension ($f_z = 5$) and manipulate the system configuration ($f_z = 9$), or an attacker can read memory locations ($f_z = 14$) and gain sensitive information ($f_z = 10$). In other words, the actions or the impact can be used interchangeably for both classes. On the other hand, when the commonality is weak, a single action or impact can imply two different concepts. For example, when an attacker gains sensitive information, this can be the list of system files ($f_z = 10$), and in the meantime, it can be the plain-text password ($f_z = 11$). Notably, action or impact refers to the same concept using a different language in strong commonality, but in weak commonality, action or impact refers to a distinct concept using the same language. According to the table, types 13 and 14 are the children of type 12 (inheritance), type 10 and 11, and 10 and 12 have strong commonality dependency, type 5 and 9 also have strong commonality, and type 10 and 11 have weak commonality dependency. Since types 13 and 14 are inherited from class 12, they have strong commonality with classes 10 and 11 as well. Such dependencies are critical in corpus generation, dataset creation, and also in model training and evaluation. To summarize, if f_z and f'_z represent two functionalities and $R(f_z, f'_z)$ demonstrates the dependency between them:

$$R(f_z, f_{z'}) = \textit{Inher} \quad \textbf{IF } f_{z'} \text{ is the inheritor of } f_z \tag{5.1}$$

$$R(f_z, f_{z'}) = \textit{Strong} \quad \textbf{IF } f_z \text{ and } f_{z'} \text{ have strong commonality dependency} \tag{5.2}$$

$$R(f_z, f_{z'}) = \textit{Weak} \quad \textbf{IF } f_z \text{ and } f_{z'} \text{ have weak commonality dependency} \tag{5.3}$$

$$R(f_{z'}, f_{z''}) = \textit{Strong} \ \textbf{IF} \quad R(f_z, f_{z'}) = \textit{Inher} \quad \text{and} \quad R(f_z, f_{z''}) = \textit{Strong} \tag{5.4}$$

Each of the preceding functionality refers to a specific malicious action carried out by an attacker in order to compromise a system. Specifying the dependencies helps in better understanding and processing the main concept of each functionality leading to a strategic text extraction and annotation, discussed in the next section.

5.5.2 EXTRACTOR

EXTRACTOR [50] is an off-the-shelf tool that leverages SRL to extract threat actions from the technical text reports in the form of a graph. SRL can assign semantic labels to phrases and words in a sentence, where each label specifies the semantic role that each phrase or word plays in the sentence in association with the predicate or verb of the sentence. In SRL, the tags assigned to sentence components are called arguments (denoted by ARG). Table 5.3 shows these arguments and the definition based on PropBank [51].

EXTRACTOR operates by performing different rounds of transformations on the text to bring it from a highly complex and potentially ambiguous form to a simpler form. This simplified text is further processed to obtain a provenance graph that can be successfully used for threat detection. Fig. 5.4 shows the four major components that the EXTRACTOR is composed of including text normalization, resolution, summarization, and graph generation.

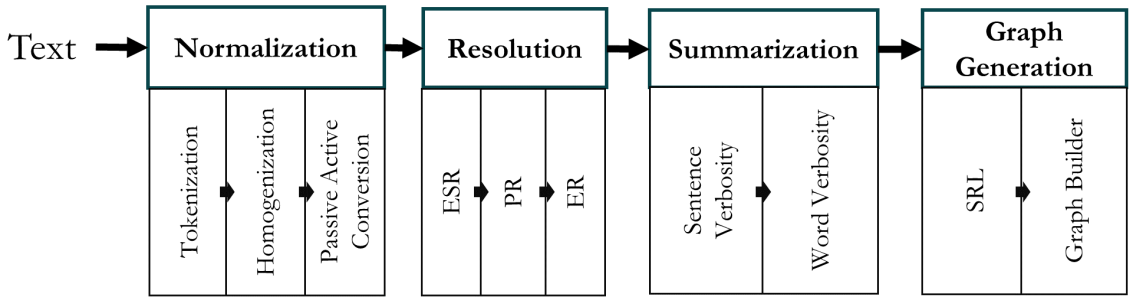


Figure 5.4: The overview of EXTRACTOR framework

Similar works mostly consider limited resources, such as sample malware or limited types of security reports, to extract threat actions. On the other hand, they focused on exchanging IOCs[4] than describing how those IOCs are connected and how the attacks behave. The main advantage of EXTRACTOR compared to similar work is that it focuses on extracting the attack behavior and captures system-level causality

[4]Indicator of Compromise

in the form of provenance graphs and utilizes the public CTI reports to convert raw text into actionable knowledge. Furthermore, the different rounds of transformations enable the EXTRACTOR to deliver clean, concise, and fine-grain threat actions that can be used directly to represent functionalities.

Normalization is responsible for an initial round of sentence simplification and transformation to a canonical form by breaking long and complex sentences into shorter sentences, which is easier to process. Normalization is comprised of tokenization, homogenization, and conversion. These steps perform the detection of sentence boundaries, word homogenization in which multiple textual representations of the same concept are replaced by the same textual representation (e.g., C2, C&C, and Command and Control are different representations of the same entity, while verbs like stores, saves may represent an action that corresponds to a write system call), and passive-to-active verb conversion.

Resolution resolves ambiguities in the sentences reconciling implicit references that refer to the same entity into the actual referent. This phase comprises three components as Ellipsis Subject Resolution (ESR), Pronoun Resolution (PR), and Entity Resolution (ER).

Ellipsis subject is a linguistic structure where a sentence's subject is not present that confuse the state-of-the-art NLP toolkits, thus resulting in the loss of the narrative sequence and the story relationships. ESR module utilizes POS and DP to detect target sentences, builds a list of candidate subjects among the entities appearing in the sentences preceding the current sentence, and then picks the most probable candidates from the list based on the distance of that candidate from the sentence with the missing subject.

PR is the process by which pronouns are mapped and substituted to the antecedent entities that they refer to. Processing documents (building a provenance graph) without PR can result in the appearance of several nodes (i.e., pronouns) for a single

entity. To resolve pronouns, EXTRACTOR adapts a popular coreference resolution model, NeuralCoref[5].

ER is the process by which noun or verb phrases that refer to another entity inside the same sentence are substituted by that entity or are eliminated as redundant focusing on actionable entities and actions that are likely to appear in audit logs, using POS tagging and DP with domain knowledge contained in CTI nouns dictionary or in a corpus of common phrases. For examples, phrases like *"tries to open* and *"makes the modification"* will be converted to *"open"* and *"modify"*, respectively, using this component. Summarization removes the portion of text that is not strictly related to the attack behavior, and that cannot be observed in the logs. Ideally, only the sentences that describe actions that may be observed in the audit logs should be preserved. For summarization, EXTRACTOR takes advantage of two-step approach to deal with sentence verbosity and word verbosity using BERT classifier and BiLSTM network. For sentence verbosity, they labeled $8,000$ threat sentences under two classes of productive and nonproductive, and trained BERT on this set. Word verbosity removes unnecessary words from the productive sentences that it receives as input from BERT using BiLSTM model and a word remover component. After a sentence is processed by a BiLSTM network, its components are tagged as *Agent*, *Patient*, and *Action*, and other types of arguments. For this component, EXTRACTOR utilizes System Entity Extractor (SEE) in which, a sentence component that is tagged for removal will be removed if it does not contain any entities that can be generated by the rules of the SEE component.

Finally, Graph Generation is responsible for resolving the temporal and causal order among the events in the text and for building the final provenance graph (this component addresses the Relationships Extraction challenge)

Some of the EXTRACTOR components may be assisted by a set of dictionaries

[5]https://github.com/huggingface/neuralcoref

that contain terms related to CTI language (relying on domain-specific dictionaries of concepts is a common approach in many knowledge-based NLP systems.

In particular, EXTRACTOR uses two dictionaries. First, a system call synonym dictionary, which contains verbs representing system calls (e.g., write, fork) and their corresponding synonyms. These synonyms represent the possible verbs that can be used in CTI reports and very likely refer to a system call. Second, the CTI nouns dictionary contains noun phrases commonly used in CTI reports, as well as different textual representations of the same concept. The former contains 87 verbs representing system calls, while the latter holds over 1112 common noun phrases in the CTI report.

5.5.3 SVO Extraction Framework

Functionalities can be divided into two categories. The first group describes "actions" that correspond to the specific exploitation. For example, *read file, create an account, and install app* specify the exact action the attacker will take after exploiting the vulnerability. The second functionality group, on the other hand, refers to the exploit's "impact" rather than its "action" in terms of the attack's final outcome. For example, this category includes functionalities that describe memory errors such as buffer errors.

For the purpose of extracting statements describing action functionalities, we represent each functionality with a set of *<Subject> <Verb> <Object>* or SVOs. The $< Subject >$ represents any type of adversary (e.g., attacker, hacker, unauthorized user, etc.), the $< Verb >$ refers to a verb (e.g., read, write, modify, etc.), and $< Object >$ is any type of cyber object that follows the verb (e.g., file, account, information, etc.). To be more precise, based on arguments in SRL, we define the *<Subject> <Verb> <Object>* sequence as $SVO = <ARG0> <V> <ARG1>$.

Let's define an argument dictionary consisting of subjects (ARG0) D_{subj}, verbs (V) D_{verb}, and objects (ARG1) D_{obj} that have been manually extracted and annotated

from CVE reports. The full corpus of verbs and objects assigned to each functionality is depicted in Table C.1 in Appendix A. If D_{subj} ={attacker, adversary, hacker, unauthorized user, unauthenticated user} is a constant dictionary of particular malicious actors, let f_z denote the functionality with index z, and $V_z \in D_{verb}$ and $O_z \in D_{obj}$ be the set of verbs (V) and objects (ARG1) associated with functionality f_z. Both of these sets are extracted based on the relevancy to each functionality and the commonality of words in CVE reports based on expert knowledge. Therefore, given the list of annotated verbs and objects, for each functionality f_z, we utilize EXTRACTOR to extract every possible statement in terms of SVOs from CVE reports that contain $verbs \in V_z$ and $Objects \in O_z$ to generate the initially labeled dataset. Thus, we represent each functionality f_z as the union of extracted SVOs as follows:

$$S(f_z) = \bigcup S_i^z V_j^z O_k^z \ \textit{where}\ S_i^z \in D_{subj}, V_j^z \in D_{verb}, O_k^z \in D_{obj} \tag{5.5}$$

In Eq. 5.5,$z \in \{1, .., 16\}$ is denoted as the functionality index. Additionally, S_i^z, V_j^z, and O_k^z correspond to subject, verb, and object associated with functionality f_z, respectively. $S_i^z V_j^z O_k^z$ also represent a SVO extracted by EXTRACTOR. Table 5.6 shows a few examples of SVOs extracted for four functionalities. Such statements

Table 5.5: Example of extracted SVOs for four functionalities

f_z	**SVOs**
Create Account	*- remote attackers create new accounts* *- unauthenticated users create accounts with arbitrary roles*
Read Files	*- remote attacker read arbitrary files* *- attackers view arbitrary files on the system*
Change Ownership	*- remote attackers modify permission field* *- unauthenticated user changes the ownership of the files*
Install App	*- unauthenticated, remote attacker install additional jee applications* *- attacker place a malicious dll file*

provide a plain and simple representation of functionalities with respect to particular actions and objects. For example, SVOs such as *remote attacker read arbitrary files* which represent functionality "Read File" involving annotated verbs and objects frequently appear in CVE reports that clearly describe any form of file (arbitrary, dll, txt, etc.) reading (read, open, access, etc.) in CVE reports that directly reflect the attacker's main action. However, this form of simple SVO extraction may not adequately describe impact functionalities or may not be quite typical for all functionalities. For example, both "*attackers read memory*" and "*attackers read memory that cause buffer over-read condition*" phrases providing actions addressing similar behavior. The former conforms to the simple format of SVOs we discussed above, whereas the latter is a bit different and provides more granular information regarding the impact of "buffer over-read" contrition. This form of threat representations are quite prevalent in CVE report format that presents a primary action or a fault in the system followed by an impact as *<ACTION/FAULT> CAUSES <IMPACT>*. We refer to this as a causal link in which, an action causes or leads to an impact, which is important for describing impact functionalities. In addition, defining such link can help to differentiate one class from another one. For example, reading memory by attacker refers to to the $f_6 = Read\ From\ Memory$, but when this action causes buffer issues, it infers the f_9 known as *Memory Read (Memory Buffer Errors)*.

We establish new sets of verbs and objects to extract SVOs related to impact functionalities, similar to the earlier approach for extracting SVOs associated with action functionalities. Let $D'_{verb} = \{\text{cause, lead, result}\}$ and D'_{Obj}, which depicted in Table C.2 in Appendix A, represent another set of annotated verbs and objects corresponding to causal link, respectively.

If f_z represents the functionality with index z, we define a new set of objects $O'_z \in D'_{obj}$ associated with f_z. Likewise, for those classes that have this causal link, we extract the SVOs utilizing EXTRACTOR. Table C.2 in Appendix A shows the

extracted objects to represent the causal links in three functionalities. Similar to Eq. 5.5, we represent each functionality f_z as the union of extracted SVOs as follows:

$$S'(f_z) = \bigcup S'^z_i V'^z_j O'^z_k \ \ where\ S'^z_i \in D_{subj}, V'^z_j \in D'_{verb}, O'^z_k \in D'_{obj} \tag{5.6}$$

Therefore, every functionality f_z is represented by the union of two sets of SVOs as $S(f_z) \cup S'(f_z)$.

It may appear ad hoc and biased to provide a specialized corpus and strict rules for extracting SVOs to represent functionalities. However, in the absence of sufficient ground truth data, we seek to make such representations as accurate as possible for training purposes. Notably, our objective is not to collect all possible representations. We collect as much data as possible and then strategically train the model to learn the connection between extracted data and an unseen text. This mechanism (see Section 5.6), especially in the absence of a standard training dataset, assists in highlighting the core content of the target functionality and defining the surrounding context to capture the semantic relationship between content and context, which can help to recognize more complicated actions and implicit patterns that the content by itself does not describe.

5.6 Methodology

We retrieved a list of SVOs for each functionality in the previous section, with each SVO associated with a CVE report. Our goal is to predict the functionality of a certain action (for example, *read arbitrary file*) based on the surrounding context.

We use SecureBERT's capacity to obtain the relationship between two phrases and leverage it to improve the text classification with document-level contextual information. Therefore, we develop a model that takes two inputs, "content" and "context," and returns the content's corresponding functionality. In essence, the term "content" refers to a typically short text that describes a particular action (in this case, func-

tionality) with no or minimal noise. Meanwhile, "context" implies a longer text which includes the content as well as additional discussions about related or similar notions.

As model input, we concatenate the content X and context D into a text sequence [<[CLS]> X <SEP> D <SEP>]. Then, in a mini-batch, pad each text sequence to M tokens, where M is the batch's maximum length. After that, the vector [CLS] is fed into a single-layer neural network with N output neurons, where N denotes the total number of functionalities. We begin with a pre-trained SecureBERT model and fine-tune it using cross-entropy loss.

To develop such a model, we would first establish a dataset in which each sample is a text pair, and then design the classification layer utilizing SecureBERT to classify the text pair into a functionality.

Dataset Creation

In Section 5.5.3 we introduced a framework to extract SVOs to represent the functionalities. To train the model, these SVOs must be structured in a specified format and paired with another relevant text. As previously noted, the objective is to build a model that can predict the functionality corresponding to a content (SVO) within a context. In other words, the same action may address multiple functionalities; thus, we aim to train a model that can identify the correlation between a particular action (content) and the longer text (context) which includes more information about the action to deliver the correct functionality to which the action refers. Hence, it is crucial to strategically pair the SVOs with a relevant text, in order to maximize training performance.

To capture different types of information, we link each SVO with three types of contexts in different stages for creating the training dataset. It worth noting that, for each functionality in all three stages, we generate a maximum τ_x number of pairs to avoid oversampling and the imbalanced data problem.

First, for each f_z, we generate τ_1 number of samples by randomly pairing the SVOs belong to f_z. This helps the model to learn the main concept of a functionality by observing different SVO representations associated with a functionality.

NOTE: Causal SVOs (Table C.2 in the Appendix) can be used as is in the content part (e.g., *attacker cause buffer overread condition*). However, for f_{12} (Memory Read) and f_{16} (Memory Modification), the CVE description of the non-causal SVO must contain the causal objects to avoid class conflicts. For example, "*attackers read memory*" can be associated with both f_{12} (Read From Memory) and f_{13} (Memory Read). This SVO would get f_9 label if the CVE description associated with f_{13} contains an object in o'_j. In other words, we look for predefined objects defined for "Memory Read" within the CVE description and if found, the SVO "*attackers read memory*" will be assigned to f_9, otherwise, it will be labeled as f_{12}.

Second, for each f_z, we generate τ_2 number of samples by randomly pairing the SVOs (content) belonging to f_z and a positive sentence (context) in the manually created dataset. Positive refers to the sentence we labeled as f_z in the manual dataset. This helps the model to learn the relevancy of the content within the context with an extended vocabulary.

Third, for each f_z, we generate τ_3 number of samples by pairing the SVOs (content) with the corresponding full CVE description (context). This helps the model to learn the content of functionality within a noisy context.

Model Design –

We use the pretrained SecureBERT as a baseline model and a add a single layer neural network on top of it as classification layer. This model takes both content and context separated by special token [*SEP*] as input and trains the model to minimize the target class prediction error.

Unlike the standard way of text classification which models typically take only one

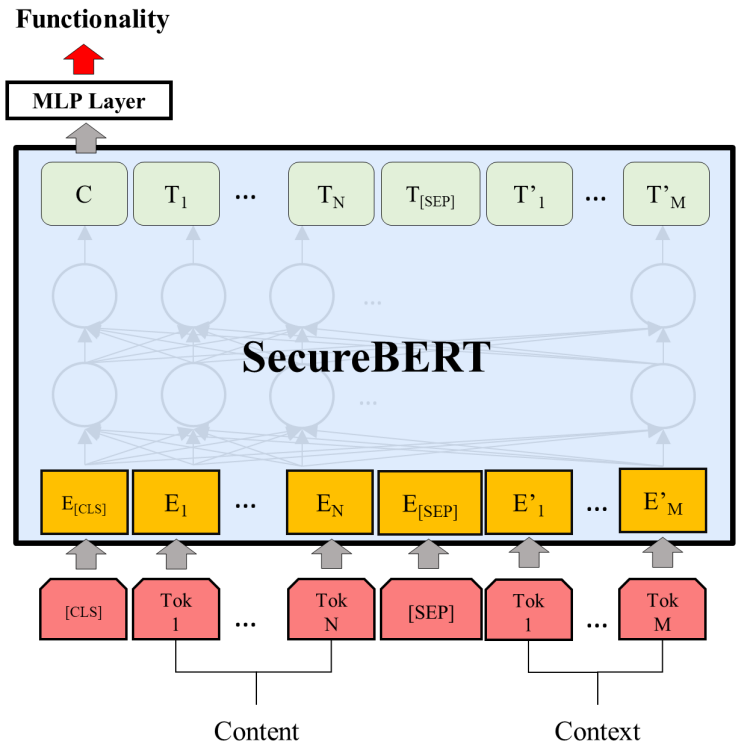


Figure 5.5: The model architecture for classifying CVEs to functionalities

input, our model is designed to take two inputs due to the specific characteristics of this classification problem. First, there is no labeled dataset from which to train such a model, and the extracted SVOs to represent functionalities are confined to the established guidelines. The context, which is a longer and more complicated text, may provide unseen yet relevant information about the extracted content (SVO) that the defined rules cannot capture. This creates a semantic link between the content and the unseen information, which greatly improves the model's learning and helps in better generalization. In addition, functionalities are sometimes interdependent, and SVOs require associated context in order to be identified correctly. In other words, two functionalities may share the same SVO, and the correct functionality for that SVO cannot be recognized unless it is assessed inside the context. For example, Table 5.6 represents two SVOs as contents extracted from the given context. The statements "*attackers read arbitrary kernel memory*" and "*attackers read kernel mem-*

Table 5.6: An example of contents and context as two inputs to the classification model.

Contents
- attackers read arbitrary kernel memory - attackers read kernel memory
Context
Linux kernel does not perform certain required *access_ ok* checks, which allows **attackers** to **read arbitrary kernel memory** on 64-bit systems and cause a denial of service and possibly **read kernel memory on 32-bit systems**.

ory" represent both "Read From Memory" and "Memory Read (Memory Errors)" if used individually. However, when such content appears with the statement "*cause a denial of service*" within the context, it clearly refers to the "Memory Read (Memory Errors)" as the corresponding functionality for the SVO. In the meantime, the context describes the underlying weakness, which is improper access check by stating "*does not perform certain required access_ ok checks*", which leads to the functionality "Memory Read (Memory Errors)". Therefore, the model can establish a semantic link between the provided weakness and the functionality during the training, that can be used as the potential indicator for predicting unseen samples in the future.

The question that may arise is why not pass only the context to the model in a standard way and ignore the content input. The justification is that contexts often include several threat-related statements, and we extract functionality-related SVOs from any report using the strategy we used to construct the dataset, thus we have no notion about other potential functionality representations in that report. As a result, the label(s) cannot be properly assigned to the report.

5.7 Evaluation

The CVE to functionality model has been trained based on content-context relationship. In other words, since there is no labeled dataset available, we had to collect and annotate data throughout pre-defined rules and an off-the-shelf tool. Therefore, the augmented dataset is imbalanced and the annotated data consists of CVE descrip-

tions and a small portion of external text. As the main objective of this model is to map specific actions to functionalities, and not classifying any random text, our model is expected to get two inputs, the content and the context, and expected to return a classification score of each content-context pair corresponding to functionalities. It worth noting that, each content in our training dataset addresses only one functionality while the context may included more, and our objective is to emphasize on the given content within the context to predict the correct functionality. Fig. 5.6 shows

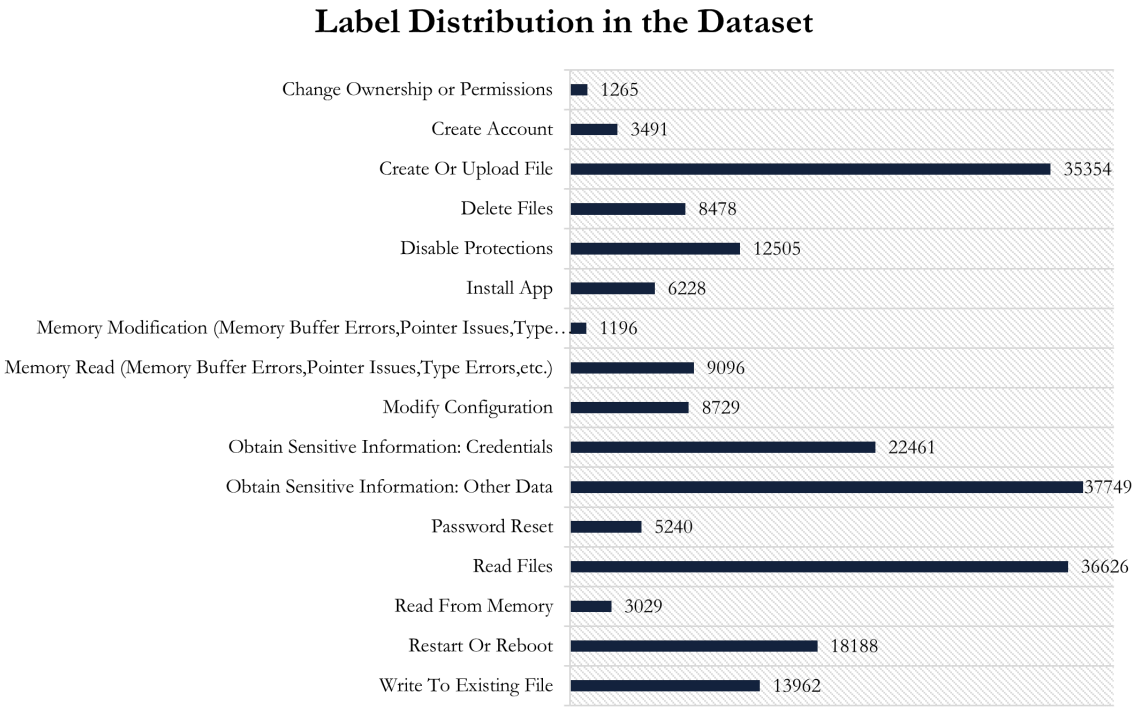


Figure 5.6: Data distribution in CVE to functionality classification dataset

the label distribution in our generated dataset that contains 223,597 content-context pair of sixteen different functionalities. As mentioned earlier, we have conducted statistical text analysis on CVE descriptions to extract commonly used terminologies in terms of subject, verbs, and objects to define rules for SVO extraction. We manually go through all the common terms and assign them to the related functionalities. In order to expand the corpus beyond the CVE descriptions, we also run the EXTRACTOR on external resources other than CVE descriptions, such as security advisories

Table 5.7: The performance of the CVE to functionality classification model using all testing dataset.

	Dataset	
Metric	$TS1$	$TS2$
Accuracy	0.981	0.983
Precision (Micro)	0.981	0.983
Recall (Micro)	0.981	0.982
F1-Score (Micro)	0.981	0.983
Precision (Macro)	0.947	0.979
Recall (Macro)	0.965	0.976
F1-Score (Macro)	0.954	0.975

and security reports from NIST, NVD, MITRE, and other vendors such as Microsoft, RedHat, Apple, etc., to extract 10,000 content-context pairs. In addition, we have also annotated 1,098 SVOs manually relying on expert knowledge. We took the 75% of data for training and the remaining 25% for testing, called $TS1$. In addition to the 25% testing dataset, we have also manually labeled 494 CVEs by extracting their content-context pairs as another testing dataset, called $TS2$. In this dataset, contents are one or more SVOs extracted from CVE descriptions and the context is the entire CVE description.

For this multiclass classification task, we use Binary Cross Entropy (BCE) as the loss function. We use 2 epochs with the Adam optimizer and an initial learning rate of $1e-5$. The model returns 16 classification scores for each input pair corresponding to each functionality, with the highest score considered the model's final prediction.

Table 5.7 shows the performance evaluation of the model on both testing datasets. According to the results, our proposed model shows high performance in predicting the correct functionalities corresponding to each input content within the given context, in both $TS1$ and $TS2$ testing datasets.

5.8 Discussions on Model Performance

Based on the confusion matrices depicted in Table 5.8 and 5.9, the model shows a high performance in predicting the correct functionality despite the existing imbal-

Table 5.8: The confusion matrix of CVE to functionality classification model using all testing dataset (*TS1 dataset*).

Correct Values \ Predicted Values	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16
f1	337	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f2	0	7221	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f3	0	0	1013	0	0	0	0	0	0	0	0	0	0	0	0	0
f4	0	0	0	1572	0	0	0	0	28	0	0	0	0	0	0	0
f5	0	0	0	0	725	0	0	0	0	0	0	0	0	0	0	0
f6	0	0	0	0	0	115	0	0	0	0	0	0	0	1	0	0
f7	0	0	0	0	0	0	663	0	1	0	2	0	0	0	0	0
f8	0	0	0	0	0	0	0	159	0	0	0	0	0	0	0	1
f9	0	0	0	0	0	0	64	0	867	0	2	0	0	0	0	0
f10	0	0	0	0	0	0	0	0	0	9341	181	0	1	1	0	0
f11	0	0	0	0	0	0	0	0	0	24	3681	0	0	0	0	1
f12	0	0	0	0	0	0	0	0	0	0	0	278	0	75	0	0
f13	0	0	0	0	0	0	0	0	0	0	34	109	9297	0	0	0
f14	0	0	0	0	0	0	0	0	0	0	0	171	0	905	0	0
f15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2422	0
f16	0	0	0	0	0	0	64	0	0	0	1	0	0	0	0	1988

Table 5.9: The confusion matrix of CVE to functionality classification model using 494 pairs of content and CVE descriptions as context (*TS2 dataset*).

Correct Value \ Predicted Values	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16
f1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f2	0	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f3	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0
f4	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0
f5	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0
f6	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0
f7	0	0	0	0	0	0	6	0	0	0	1	0	0	0	0	0
f8	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0
f9	0	0	0	0	0	0	0	0	10	0	1	0	0	0	0	0
f10	0	0	0	0	0	0	0	0	0	151	3	0	0	1	0	0
f11	0	0	0	0	0	0	0	0	0	0	27	0	0	0	0	0
f12	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0
f13	0	0	0	0	0	0	0	0	0	0	0	0	107	0	0	0
f14	0	0	0	0	0	0	0	0	0	0	0	2	0	25	0	0
f15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	37	0
f16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22

ance data problem in our dataset. However, there are some confusions, particularly in a few classes which share some sorts of dependencies as weak/strong commonality or inheritance. functionality *Read From Memory* (f_{12}), *Read Files* (f_{13}), and *Memory Read (Memory Errors)* (f_{14}) which share inheritance dependencies, return the highest confusion in the prediction. This confusion mostly happens in dataset $TS1$ in which, input pairs may share ambiguous meanings with multiple intentions. For example, the combination of some texts such as "*local user read arbitrary kernel memory location.*", "*local users, including low integrity processes, read and write to arbitrary memory locations.*", "*local user read arbitrary memory*" to generate content-context pairs can refer to any "Read" action from the memory. Such texts as a context if provided with a context potentially about the impact or further intuitions can lead to better prediction. Similarly, *Obtain Sensitive Information - Other Data* (f_{10}) and *Obtain Sensitive Information - Credentials* (f_{13}) can also share similar properties in some inputs leading to confusion. For example, *sensitive data* or "sensitive information" in "*remote attacker obtain sensitive data*" or "*local attacker obtain sensitive information*" may refer to "credentials" or other types of information such as "directory/file names" or "browser history", etc., so they must be followed by an informative context such as "*XXX could allow a local attacker to obtain sensitive information, caused by plain text* ***user account passwords*** *potentially being stored in the browser's application command history. By accessing browser history, an attacker could exploit this vulnerability to obtain other* ***user accounts' passwords***." for better prediction. The evaluation on dataset $TS2$ which indeed contains precise context (entire CVE descriptions) that is depicted in Fig. 5.9 shows higher performance in functionalities whose characteristics have some types of dependencies.

In real-world applications, the "content" may not always be available. In practice, cybersecurity researchers or software vendors often only have access to a security report (e.g., a CVE description) and intend to classify it to functionalities. Without

requiring the second input, our proposed model can take such security reports and return the corresponding scores to each of the sixteen functionalities. In fact, the trained model has been processed in order to capture the contextual representation of the input document in accordance with any of the functionalities. It is worth noting that, delivering all functionalities associated with a document, that can include more than one functionality, implies a multi-label multiclass classification approach. However, in the absence of a standard multi-label dataset, it is not feasible to train the model in such a way. Thus, as a proof-of-concept, we evaluated the model on 66 CVEs descriptions that are mapped to one or more functionalities by MITRE in the guideline. Table C.3 in the Appendix shows the model's top K predictions where K equals the total number of predictions until all the correct classes are predicted for each individual CVE description. In other words, if a CVE is associated with m functionalities, we show at least m top predictions. If all returned predictions exactly match the ground truth, we do not provide any further prediction, and hence, $K = m$. However, if all the correct classes are not covered in the top m predictions, we show another m' predictions until all correct classes are covered, and therefore $K = m + m'$. Based on the results, in 58 out of 66 data samples, the top K model predictions are equal to the number of correct classes ($K = m$ and $m' = 0$) implying an 87.88% overall hit rate. In 6 data samples, the model returns one extra prediction to cover all correct classes (($K = m + m'$ and $m' = 1$) which represents a cumulative 96.97% hit rate with one false positive. Overall, the model shows a 100% hit rate in the top 5 ($K = 5$) predictions for all test data.

The promising performance confirms that the CVE language is effectively assessed in the dataset creation and that the model appropriately captured the textual features and semantic relationship between the text and the functionalities during training.

Limitations and Future Works

Although the model predicts correct functionalities based on both content-context

and context-only inputs, it is nevertheless constrained in some ways.

This model is mainly trained on text retrieved from the CVE language. Despite the level of threat action understanding gained from CVE language, it is unclear how this would perform on other corpora such as CTI and security reports due to a lack of testing data for evaluation. Similarly, more data is required to evaluate the model on context-only input. In addition, still there are some uncertainties in predicting functionalities with dependencies which is as result of data limitation.

In the future, we wish to improve the data extraction so that we can annotate more data, including multi-label instances, to improve the model's capacity to do multi-label classification with a single input. In addition, we intend to update and improve the functionality classes defined in the guideline in order to eliminate dependencies and, as a result, offer a more robust classification model.

5.9 Conclusions and Future Works

In this work, we introduced a system to automatically classify CVEs to functionalities that an attacker gains access to after an exploit. This is a unique problem that the MITRE guideline has recently addressed [6], therefore there is no similar research working on it and, as a result, no standard dataset to develop a classification model. To address this issue, we conducted a statistical analysis of CVE descriptions and established a semi-automated framework for extracting labeled data and creating a dataset. Next, on top of the SecureBERT, we built a novel model architecture that receives two inputs, called content and context, and returns the associated functionality. This novel design helps in capturing the semantic relationships between the inputs and, therefore, identifying implicit text features that are not present in the generated dataset. As a result, it supports bypassing the limitations in the extracted data caused by specific restrictive rules we created manually.

This work offers substantial contributions to the fields of cyber threat intelligence,

[6] October 2021

threat hunting, and vulnerability assessment, as well as helping in linking the CVEs to MITRE techniques and accordingly critical security controls, which is an essential connection toward automating the course of action retrieval. In this study, we made the following contributions:

1. Create a framework to automatically extract and annotate cybersecurity textual data.

2. Propose a new dataset generation strategy that can handle data shortage for training domain-specific classification tasks

3. Propose a novel model design in order to automatically detect attack behaviors within a particular context, and classify such behaviors to functionalities.

4. Provide a comprehensive evaluation of the proposed method and show its effectiveness compared with different other models.

5. Discuss the advantages and limitations of the work and propose future plans to better generalize the model and improve the performance.

CHAPTER 6: Neural Information Retrieval (IR) Model for Retrieving Course of Defense Actions for CVEs

System security is jeopardized as a result of software vulnerabilities. Patching can usually resolve vulnerabilities, however, patches are not always available, or they may not always recommended owing to high overhead and potential service outages. The existing security strategies, vulnerability detection, and mitigation approaches are not intelligent, automated, self-managed, and not competent to combat vulnerabilities and security threats and provide a secured self-managed software environment to the organizations. Hence, there is a strong need to devise an intelligent and automated approach to optimize security and prevent the occurrence of vulnerabilities or mitigate the vulnerabilities.

To automatically mitigate vulnerabilities, a thorough assessment of underlying weakness(es), impacts, techniques, and tactics is required to understand the threats introduced by CVEs. Meanwhile, there is no labeled dataset for mapping CVEs to defense measures. In previous chapters, we developed SecureBERT as a cybersecurity language model and used it to enhance and enrich CVEs. We classified CVEs to CWEs, vulnerability types, and functionalities using multiple models built on top of SecureBERT. These classification models not only help in CVE awareness, but also establish links between CVEs, cybersecurity standards such as CWEs and MITRE ATT&CK, and accordingly their mitigation strategies.

In this chapter, we use the mitigations provided by the aforementioned cybersecurity standards to build a labeled dataset and design an information retrieval framework on top of SecureBERT to automatically infer the appropriate course of defense actions for CVEs.

6.1 Introduction

Nowadays, software systems are the lifeblood and most visible component of practically all contemporary and complicated systems. As a result, several firms rely heavily on interdependent and networked software systems to make business choices. These software applications are used to manage and regulate the operations and performance of enterprises. Almost all industries have risen their business horizons, improved their performance, and earned substantial returns through the use of the software. However, many organizations have suffered major financial and reputational losses as a result of security breaches, weak security standards, cybersecurity attacks, and induced software vulnerabilities in old and contemporary systems. The exploitation of security vulnerabilities is a severe threat to the stability of software systems and the safety of the data handled by them, as proven by new data breaches, ransomware attacks, and large disruptions of critical systems. Despite the software engineering community's ongoing efforts to improve software quality and security through secure coding guidelines, software testing, and various forms of code review, CVE records show an increase in the number of vulnerabilities discovered and disclosed each day. Although the patching would fully solve the security glitch, it is not always the choice since patches may not be available immediately exposing the system to threats for a long period, or it may not be recommended owing to high overhead and potential service outages the patch application may cause. Therefore, in response to the ever-increasing threat of cyber-attacks on critical cyber infrastructure, such firms are focusing on expanding their cybersecurity knowledge base to mitigate the growing threats. Vulnerability mitigation is defined as an attempt to minimize the impact and risk of a vulnerability without completely eliminating it. As a result, mitigating a vulnerability is a temporary yet effective solution for system protection.

Detail information about mitigation techniques might be difficult to locate and is usually only reported on the vendor or third-party websites, if available. However,

when vulnerabilities are described as higher abstraction cybersecurity standards such as weaknesses (CWEs) and techniques (MITRE ATT&CK), it is possible to identify threats across several perspectives, connect them to security measures, and mitigate them. Many current practices perform this manually, relying on expert knowledge, which is highly inefficient, costly, and prone to error. As a result, an automated system capable of precisely analyzing CVEs, identifying threat behaviors and impacts, and quickly delivering mitigation strategies is in high demand.

Information retrieval is the study of looking for information within a document or searching for the documents themselves (IR). The term "document" in this context refers to any sort of data, including text, image, video, or audio. The Google search engine and Microsoft Bing are two well-known examples of IR systems, with billions of users using such search engines to find the information they seek. In many situations, they strike gold and immediately find what they're looking for; in others, it takes more time and effort; and sometimes, they never get a satisfying answer. Therefore, enhancing the IR system's performance is critical.

The recent state-of-the-art IR models have been applied in numerous cybersecurity automation. In some circumstances, the objective is to extract some form of information from random corpora, such as cybersecurity-related data (e.g., retrieving documents relevant to InfoStealer), there is data available on the web, or gathering data is not expensive. However, for more specialized and content-sensitive tasks, such as the course of defense action retrieval, one of the most major barriers is the collection of labeled data. In addition, the model should be specially designed to fill the semantic gap that exists between the input and the output.

In this work, we aim to build an IR model utilizing pretrained SecureBERT that is able to automatically retrieve course of defense actions (CoA) in form of mitigation strategies and security controls associated with an input CVE description. CoAs are not low-level advisory to fix the issues, instead, they are high-level actions targeting

the root causes of threats, such as weakness and techniques, which are extremely helpful for cyber threat intelligence and targeted threat mitigation. Here, we train a proof-of-concept model to return the known course of defense actions provided by cybersecurity standards.

6.2 Related Works

Bhandari *et al.* [52] proposed a platform that can automatically identify and collect vulnerabilities. The dataset comprises low-level code and vulnerability information, such as vulnerability type and CWEs, severity level, function, and file, which makes it easier for developers to obtain data. This data has been collected from GitHub, GitLab, and Bitbucket in which, code snippets have been classified to CVE type. In this dataset, code changes are extracted at the file and method levels and then classified as code fixes, covering 5365 CVE records for 1754 open-source projects that were addressed in a total of 5495 vulnerability patching commits. In another work focusing on linking threat reports to MITRE technique and tactics, Liao *et al.* [53] leveraged NLP techniques to automatically extract compromised signatures such as botnet IPs and malware names from unstructured text to a more standardized format. Our work differs from this work because we consider classifying CVEs and threat reports into tactics and techniques an attacker employed to complete an attack. Our work focuses on the overall attacker behavior rather than just extracting tools used by the attacker. Burger *et al.* [11] classified various threat-sharing technologies on how they inter-operate. By considering the different use cases of the various threat-sharing technologies, they propose a way to unify these techniques for wider usage and adoption by security professionals.

6.3 Problem Definition

For each CVE description input, we aim to solve the problem of retrieving defensive strategies from a set of predetermined mitigation techniques supplied by cybersecurity

standards, such as CWEs, MITRE ATT&CK, and security controls.

This is a many-to-many classification task in which one CoA corresponds to multiple CVEs and one CVE is connected with one or more CoA. In other words, given a pool of CoAs, we seek to return all relevant ones for each CVE input. As CoAs can be updated by adding or modifying them, there is no fixed number of CoAs that can apply to a single CVE. Therefore, traditional classification schemes with a constant output size would be ineffective. Nevertheless, an adaptive and expandable IR model appears to be an effective choice.

Traditional IR models, such as those used in eCommerce product search, often rank documents using legacy relevance functions such as TF-IDF or BM25 [54]. These relevance functions are based on exact or "hard" token matches rather than semantic "soft" matches, are insensitive to word order and have static values rather than learning weights. Despite its simplicity, the legacy relevance function is insufficient for the fine-grained ranking of search results in practice. In contrast to the traditional methods, the recently introduced approach neural IR (NIR) [55] learns vector-space representations of both queries and documents, allowing neural models to address the problem of relevance ranking end-to-end. The NIR model learns the properties of both objects semantically, not solely based on statistical features, and adjusts the weights in such a manner that the semantic links between them are captured, leading to fine-grained retrieval results.

Given a CVE text description, we aim to design a proof-of-concept NIR model on top of the pretrained SecureBERT that returns three sets of CoAs corresponding to weaknesses, techniques, and security controls. To this end, we train a model to understand the semantic relationship between attack language (CVE description) and defense language (CoAs), with the model taking a CVE description and delivering a list of corresponding CoAs. In this model, we demonstrate the capability of retrieving relevant CoAs for CVEs using available unstructured data utilizing pre-

trained SecureBERT, allowing the proposed approach to be expanded and improved by training against any arbitrary data.

6.4 Challenges

Mapping CVEs and threat actions to defense actions has long been a difficult topic in cybersecurity. In this section, we define and explain the most significant challenges we faced while working on this work.

Semantic gap between attack and defense language

There is a terminological and semantic difference between the text addressing an attack and the text corresponding to a cybersecurity defense. The semantic gap is the difference between two descriptions of a similar or correlated topics that use different linguistic domains. In cybersecurity, a text describes either an attack or a defense, making it difficult for a machine to determine which is which. For example, the sentence "*attacker sends many requests to cause denial of services*", and the phrase "*block the IP address*" are two types of text having a relationship which the latter one *mitigates* the former text. Understanding such a relationship is a challenging task when it comes to predictive models. When such semantic representation is required, traditional text mining methods that utilize statistical techniques such as TF-IDF and word matching, may not effectively perform.Hence, it is important to employ a semantic-based approach capable of digesting and identifying such implicit relationships.

Lack of ground truth data

Detailed information about mitigation strategies might be available on vendor or third-party websites, if any. Thus, there is no centralized dataset for training a model for retrieving CoAs automatically. Instead, cybersecurity standards define high-level defense measures that, when linked to CVEs, can be employed to protect the system from the threat.

6.5 Methodology

In the absence of a labeled dataset, we leverage the link between CVEs and the cybersecurity standards to create different datasets. Then, we design a multi-model framework to return the corresponding CoA for any CVE input.

6.5.1 Data Augmentation

In the previous chapters, we classified CVEs to different cybersecurity standards, including CWEs, vulnerability types, and functionalities. Vulnerability types and functionalities are connected to common MITRE ATT&CK techniques, which are also linked to security controls via MITRE guidelines. In this regard, we establish three datasets comprising CoAs from CWEs, MITRE ATT&CK, and critical security controls (CIS CSC), corresponding to all CVEs that are directly or indirectly tied to any of these standards (through vulnerability types, functionality, and MITRE techniques). Fig. 6.1 shows the CVE connections with cybersecurity standards and those who share CoA.

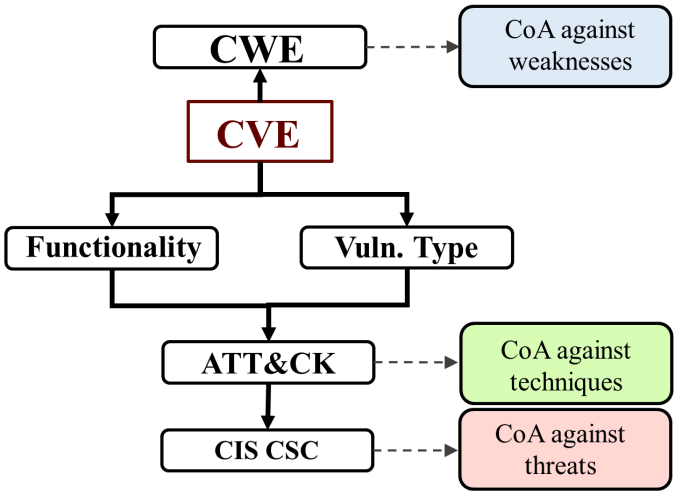


Figure 6.1: CVEs' connections and the source of course of actions.

6.5.2 Model Design

In order to retrieve the list of CoAs for a given CVE, we design a neural information retrieval (NIR) model leveraging SecureBERT. The NIR models typically take a query-document key and return the relevance score. Similarly in our proposed model, the query (e.g., a CVE description) and document (e.g., a CoA) tokens are concatenated and then separated into two segments by the special token [*SEP*], with the [*CLS*] at the beginning of the first segment. In the output, the embedding of the first token is used as a representation for the entire query-document pair and it is fed into a multi-layer perceptron (MLP) to predict the possibility of relevance. Therefore, the output size of the classification layer equals to "one", returning the relevance score. Fig. 6.2 shows the initial neural information retrieval model. The model takes two inputs in the form of query Q and document D, separated by a special token [*SEP*]. The query Q is the input addressing the attack and document D refers to defense input. The SecureBERT tokenizer returns a vector of token IDs and passes it through the transformers stack. The output of the [*CLS*] token (output C) is then used to represent the entire input text, which is passed into another MLP layer on top of the transformers stack to predict the relevance score of two inputs.

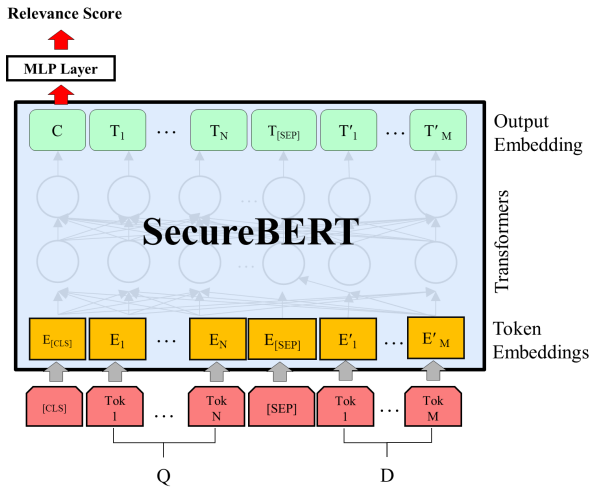


Figure 6.2: The initial neural information retrieval model.

The mission of this project is to train separate but interconnected models for determining the best set of defense actions to take in response to a CVE employing mitigation strategies provided by CWEs, MITRE ATT&CK, and security controls. Each standard's recommended CoAs may differ in structure and offer a "unique" approach to mitigating the threat. In other words, risk can be reduced by addressing the weakness or safeguarding the techniques that lead to an exploit. This uniqueness justifies the use of a separate model for each standard, so that each model offers a different set of defense actions, allowing defenders to to choose the one that works best for them.

Meanwhile, as all CoAs include defensive language which is different than CVE text as well as sharing some mutual information, the semantic and contextual representation of each model, corresponding to each standard, can help in faster and more efficient learning in other models. In other words, following the concept of "transfer learning", a model which is fine-tuned against one CoA can be used to improve the model performance of other models. Table 6.1 shows the CoAs suggested by different cybersecurity standards for particular defense strategies such as Encryption, Access Limit, and Sandboxing. In this context, CoAs may refer to similar defense actions, regardless of the original sources. For example, when a model is already trained on CoAs suggested by CWEs related to sandboxing, it would better converge when trained against similar CoAs suggested by MITRE ATT&CK or CSC. Therefore, we propose a multi-model training approach in which the model is trained in a series of interdependent sequential stages, each one building upon the preceding one. To be more concise, we begin with training the NIR model to predict CWE-level CoAs using an initial model called $SecureBERT_{CWE-Defense}$. This model accepts as inputs CVE or CWE descriptions (attacks) and CWE mitigations (defense input). This model is capable of associating CVEs with their relevant CWE-level CoAs. The following phase involves further fine-tuning the initial model using CVE, CWE, and MITRE

Table 6.1: CoAs offered by different cybersecurity standards for different defensive methods.

Strategy	CoAs
Encryption	**CWE**: Encrypt the data with a reliable encryption scheme before transmitting.
	MITRE ATT&CK: Ensure that all wired and/or wireless traffic is encrypted appropriately. Use best practices for authentication protocols, such as Kerberos, and ensure web traffic that may contain credentials is protected by SSL/TLS.
	CSC: Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).
Access Limit	**CWE**: Limit Content Provider permissions (read/write) as appropriate.
	MITRE ATT&CK: Use least privilege for service accounts will limit what permissions the exploited process gets on the rest of the system.
	CSC: Configure data access control lists based on a user\'s need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.'
Sandboxing	**CWE**: Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system.
	MITRE ATT&CK: Make it difficult for adversaries to advance their operation through exploitation of undiscovered or unpatched vulnerabilities by using sandboxing.
	CSC: Deploy and maintain email server anti-malware protections, such as attachment scanning and/or sandboxing.

ATT&CK technique descriptions as attack inputs and MITRE ATT&CK mitigation as defensive inputs. This model is called $SecureBERT_{ATT\&CK-Defense}$. In contrast to the previous phase, this stage fine-tunes the $SecureBERT_{CWE-Defense}$ rather than the SecureBERT. By applying this strategy, we ensure that $SecureBERT_{ATT\&CK-Defense}$ has a working knowledge of the defense language (CWE-based CoA), which favors faster and more effective convergence. Similarly in the last phase, we take the $SecureBERT_{ATT\&CK-Defense}$ as the initial model and feed it with CVE, CWE, or MITRE ATT&CK technique descriptions as attack inputs and security controls as defensive inputs and train a model called $SecureBERT_{Control-Defense}$. Fig. 6.3 shows

the overall architecture of our proposed model.

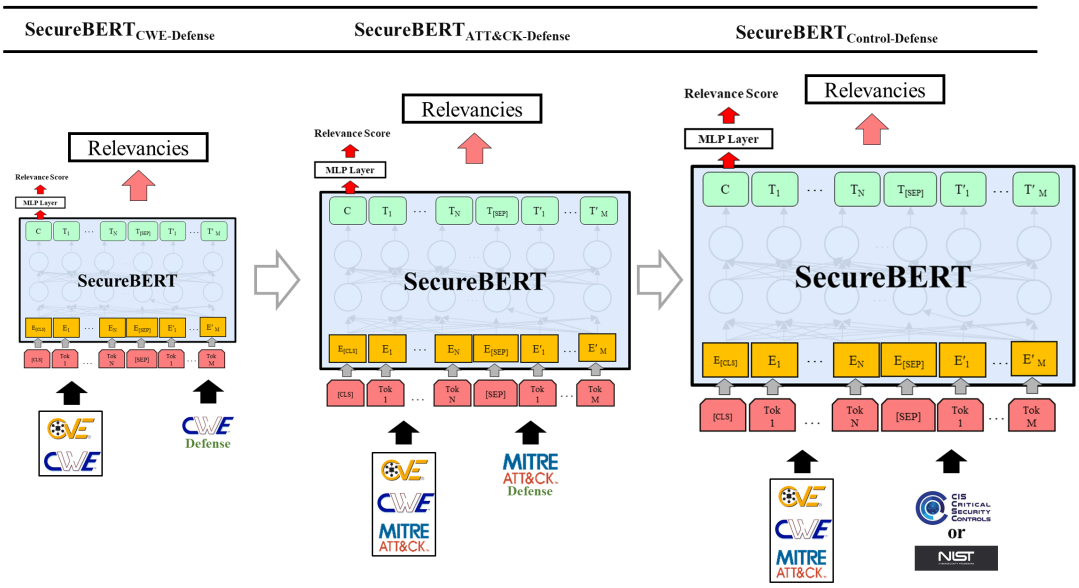


Figure 6.3: Show the model design for retrieving CoAs.

One question that may arise is why not aggregate all CoAs and train a single model that takes all CVEs and all CoAs at the same time and train it accordingly. Our method is justified for two reasons: simplicity and specificity. Dividing the problem into smaller sub-problems simplifies training and results in a more robust model. There are several CoAs to respond to a CVE, and returning all at once could be burdensome for the user in terms of analyzing, selecting, and applying multiple options. Furthermore, it may induce confusion during training since similar (in terms of language) but different (in terms of defense action) CoAs may lead to the formation of an inefficient model. Regardless of simplicity, multiple models imply multiple strategies to respond to a threat. As security controls, each model provides CoAs against a specific property of a CVE exploit, such as a weakness, technique, or defined set of standard security controls. This allows defenders to decide how to mitigate the threat based on their capabilities, assets, and needs that enables the defenders to choose and customize their desired defense approach.

6.6 Evaluation

In this section, we undertake a comparative evaluation of our proposed model's performance in retrieving CoAs for CVEs. We train three separate neural information retrieval models on top of each other, corresponding to three cybersecurity standards including CWEs, MITRE ATT&CK, and Critical Security Controls so that each returns the corresponding mitigation strategies.

To train each model, we first collect CVEs and a list of corresponding CoAs. As there is no direct link between CVEs and CoAs, we utilize the connection between CVEs and the cybersecurity standards to collect and annotate the CoAs. Thereafter, we create CVE-CoA pairs, as query-document inputs, by utilizing positive and negative sampling. Training an IR model generally entails receiving both positive (CVE and CoA are relevant) and negative (CVE and CoA are irrelevant) examples, learning the representations of the query (Q) and the document (D) based on the loss function, and then completing model training. The use of negative sampling in information retrieval and recommendation systems has a significant impact on the training of an effective model. Therefore, for each CVE, we conduct positive sampling by pairing it with all relevant CoAs and assign 1 as relevance score. Meanwhile, we employ a heuristic method, Popularity-biased Negative Sampling (PNS) [56] to produce the negative examples with the relevance score of 0. In this context, we pair each CVE with N irrelevant CoAs which we practically used the value of 15 for N.

Each cybersecurity standard provides a unique number of CoAs. As depicted in Table 6.2, for the labeled CVEs in our datasets, CWEs, MITRE ATT&CK, and CIS CSC suggest 202, 95, and 20 mitigation strategies, respectively. To train each model, there is a specific dataset, each comprises a unique number of CVEs connected the associated cybersecurity standards. Table 6.3 shows the number of CVEs exist in each training dataset as well as the total number of generated CVE-CoA pairs which is used to train the models.

Table 6.2: Shows the number of mitigations each cybersecurity standard provides for the CVEs.

Standard	**No. CoAs**
CWEs	202
MITRE Techniques	95
CIS CSC	20

Table 6.3: This table represents to number of CVEs associated with each cybersecurity standard, and shows the total number of CVE-CoA pairs generated tp train each model.

Standard	**No. CVEs**	**No. CVE-CoA Pairs**
CWEs (Vuln. Types)	55,205	1,663,517
MITRE ATT&CK	38,936	798,793
CIS CSC	32,180	535,655

We employ Binary Cross-Entropy with logits (BCEwithLogits) [1] as the loss function for this multiclass classification problem. BCEwithLogits loss combines a Sigmoid layer and the Binary Cross-Entropy (BCELoss) in one single class. It is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, it takes advantage of the log-sum-exp trick for numerical stability. We practically employ two epochs with the Adam optimizer and a $1e-5$ initial learning rate. For every input pairs, each model generates one relevance score between 0 and 1. We utilize the training dataset (Table 6.3 for training the model. During the testing, the model takes a CVE description rather than a CVE-CoA pair and returns a ranked list of CoAs with the relevant scores. we tested $SecureBERT_{CWE-Defense}$, $SecureBERT_{ATT\&CK-Defense}$, and $SecureBERT_{Control-Defense}$ with *11,041*, *7,787*, and *6,436* unseen CVE descriptions. To evaluate the model performance, we used two standard IR evaluation metrics including Mean Reciprocal Rank at K (MRR@K) and Mean Average Precision (MAP@K), where $K \in \{5, 10, 15\}$. MRR is a metric that examines, "Where is the first relevant item?" It is directly related to the binary relevance metric family. For a single query, the reciprocal rank is $\frac{1}{rank}$ where *rank* refers

[1] `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`

to the position of the highest-ranked answer. If no correct answer returned at top K, the reciprocal rank is 0. The reciprocal rank of a set of recommendations served to a single query i ($RR_i@K = \frac{1}{rank_{i,1}}$) is defined as the rank of the first relevant document among the top K. For multiple M queries, MRR@K is defined as the average of M reciprocal ranks as $MRR@K = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{rank_i}$. This approach places a strong emphasis on the first relevant element in the list. It works best for targeted searches, such as when looking for the "best" CoA. The MRR metric does not evaluate the rest of the list of retrieved documents and it focuses on a single correct CoA from the list. In our problem, it is calculated by taking any of the correct CoAs as the target answer if located in the top K, regardless of other correct answers. The MRR@K shows how well the model can distinguish between the relevant and non-relevant documents.

To measure if "all the predicted items are relevant" and if "the most relevant items ranked at the top", we compute the MAP@K. By calculating the average precision at K (AP@K) over all instances in the dataset, MAP@K determines whether all relevant items tend to be ranked highly. The average relevance scores of the top K documents returned in response to a query are used to calculate AP@K. It compares the set of top K results for each query to the set of correct (relevant) documents, i.e. a ground truth set of relevant documents for the query. Therefore,

$$AP@K = \frac{1}{N(k)}\sum_{i=1}^{K}\frac{TP_{seen}(i)}{i}$$

where TP stands for True Positives and $N(k)$ and TP_{seen} can be calculated as follows:

$$N(k) = minimum(k, TP_{total})$$

and

$$\begin{cases} 0 & i^{th} \text{ is NOT relevant} \\ TP_{seen} \quad \text{till i} & i^{th} \quad \text{is relevant} \end{cases}$$

Table 6.4: The evaluation of three proposed models for retrieving course of actions for CVEs.

Model	MRR@5	MAP@5	MRR@10	MAP@10	MRR@15	MAP@15
$SecureBERT_{CWE}$	92.13	92.76	93.56	93.88	94.19	96.13
$SecureBERT_{ATT\&CK}$	94.10	94.29	94.28	94.97	94.39	96.41
$SecureBERT_{Control}$	95.61	96.01	95.98	98.76	96.02	99.29

Table 6.4 shows the performance evaluation of all three models. According to the results, our proposed model shows a high performance in retrieving correct CoAs for CVE entries.

NIR models are often planned to retrieve relevant documents from a large number of documents and thus require a lot of training samples to perform effectively. In this problem, there are two key correlations between the CVEs and CoAs need to be recognized during the training. First, the connection between the attack and defense language and second, the connection between CVE and the cybersecurity standards which CoAs are extracted from. Detecting such correlations during the training leads to a high retrieval performance in all the models. In the meanwhile, unlike typical NIR applications, the document pool size is relatively small (202, 95 and 20 as depicted in Table 6.2) which results in less complex model and accordingly smoother training process.

Limitations

Despite the high performance of all three models, there are some potential limitations in this work.

First, the lack of labeled data leads to not cover all possible CoAs to be considered when training the models. The CVE-CoA connections for establishing the labeled dataset has been derived from the most commonly seen methods in the cybersecurity standards. However, these connections are limited to what the guideline has provided and therefore, several CWEs, MITRE ATT&CK techniques, and security controls have not been considered. Therefore, this method have potential limitations in case of facing infrequent threats. This problem can be solved by expanding the guideline

and providing CVEs with more connections to the higher abstractions.

In addition, the CoAs derived from cybersecurity standards are not CVE-specific, implying high-level mitigation plans. As noted previously, CVE-specific mitigation methods can be found on vendor or third-party websites; hence, approaches such as code-base solutions or detailed step-by-step recommendations are not offered in cybersecurity standards CoAs. The CoAs proposed by the models in this work provide high-level guidelines for reducing risk and minimizing the impact of the exploit. For the same reason, all of the returning CoAs may not apply to the given CVE. Our models provide a short list of potential mitigation measures, and the user must decide how, what, and where to respond in order to counter the exploit.

6.7 Conclusions

In this section, we introduced a multi-model framework to process the CVEs' description and retrieve the corresponding course of defense actions provided by different cybersecurity standards including CWE, MITRE ATT&CK, and Critical Security Controls (CIS CSC). In general, we make the following contributions as followsn:

- Creating a proof-of-concept neural information retrieval (NIR) model to retrieve appropriate set of course of defense actions for a CVE with respect to the CVE's associated weaknesses, techniques, and security controls.
- Addressing the existing semantic gap between attack and defense language by designing a multi-model framework.
- A thorough evaluation of the proposed model.

Conclusion

By employing AI-based technologies such as machine/deep learning and natural language processing, security systems can be trained to automatically assess, analyze, and mitigate cyberthreats. In contrast to simply sounding an alarm to alert a human security technician to take action, an automated cybersecurity system can swiftly assess, analyze, characterize, and mitigate a vulnerability. Automating security systems not only improves efficiency by reducing human error, but also boosts overall efficiency by accelerating response time and minimizing costs.

This dissertation aimed to develop a framework to automatically parse and analyze human-readable description of the CVEs in order to characterize and mitigate cybersecurity vulnerabilities. Therefore, it introduced SecureBERT, a domain-specific language model to understand the cybersecurity language, and leveraged it to (1) deploy multiple NLP models to classify CVEs to the most common higher abstraction security standards including CWEs and MITRE ATT&CK techniques to enrich CVE descriptions, (2) predict the corresponding CVSS metrics for the purpose of prioritization, and (3) retrieve the associated course of defense actions (CoA). Each section of this work presents several innovative techniques for data generation and model design, reports a full evaluation procedure to illustrate the performance, and discusses the limitations of each framework.

REFERENCES

[1] M. Corporate, "Common weakness enumaration," 2018.

[2] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," *arXiv preprint arXiv:1712.09405*, 2017.

[3] 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)*, pp. 1532–1543, 2014.

[4] P. Ranade, S. Mittal, A. Joshi, and K. Joshi, "Using deep neural networks to translate multi-lingual threat intelligence," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 238–243, IEEE, 2018.

[5] S. Mittal, A. Joshi, and T. Finin, "Cyber-all-intel: An ai for security related threat intelligence," *arXiv preprint arXiv:1905.02895*, 2019.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[8] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[9] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," *arXiv preprint arXiv:1903.10676*, 2019.

[10] A. Dalton, E. Aghaei, E. Al-Shaer, A. Bhatia, E. Castillo, Z. Cheng, S. Dhaduvai, Q. Duan, B. Hebenstreit, M. M. Islam, *et al.*, "Active defense against social engineering: The case for human language technology," in *Proceedings for the First International Workshop on Social Threats in Online Conversations: Understanding and Management*, pp. 1–8, 2020.

[11] M. S. I. Sajid, J. Wei, M. R. Alam, E. Aghaei, and E. Al-Shaer, "Dodgetron: Towards autonomous cyber deception using dynamic hybrid analysis of malware," in *2020 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, IEEE, 2020.

[12] E. Aghaei and G. Serpen, "Host-based anomaly detection using eigentraces feature extraction and one-class classification on system call trace data," *Journal of Information Assurance and Security (JIAS)*, vol. 14, no. 4, pp. 106–117, 2019.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[14] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[15] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[16] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[17] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa, "Byte pair encoding: A text compression scheme that accelerates pattern matching," 1999.

[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[19] C. Wang, K. Cho, and J. Gu, "Neural machine translation with byte-level subwords," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9154–9160, 2020.

[20] R. M. Zur, Y. Jiang, L. L. Pesce, and K. Drukker, "Noise injection for training artificial neural networks: A comparison with weight decay and early stopping," *Medical physics*, vol. 36, no. 10, pp. 4810–4818, 2009.

[21] Z. You, J. Ye, K. Li, Z. Xu, and P. Wang, "Adversarial noise layer: Regularize neural network by adding noise," in *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 909–913, IEEE, 2019.

[22] X. Liu, M. Cheng, H. Zhang, and C.-J. Hsieh, "Towards robust neural networks via random self-ensemble," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 369–385, 2018.

[23] C. M. Bishop, "Training with Noise is Equivalent to Tikhonov Regularization," *Neural Computation*, vol. 7, pp. 108–116, 01 1995.

[24] X. Li, Z. Yang, P. Guo, and J. Cheng, "An intelligent transient stability assessment framework with continual learning ability," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 12, pp. 8131–8141, 2021.

[25] H. Ahn, S. Cha, D. Lee, and T. Moon, "Uncertainty-based continual learning with adaptive regularization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[26] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[27] E. Aghaei and E. Al-Shaer, "Threatzoom: neural network for automated vulnerability mitigation," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pp. 1–3, 2019.

[28] E. Aghaei, W. Shadid, and E. Al-Shaer, "Threatzoom: Hierarchical neural network for cves to cwes classification," in *International Conference on Security and Privacy in Communication Systems*, pp. 23–41, Springer, 2020.

[29] S. K. Lim, A. O. Muis, W. Lu, and C. H. Ong, "MalwareTextDB: A database for annotated malware articles," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Vancouver, Canada), pp. 1557–1567, Association for Computational Linguistics, July 2017.

[30] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "Biobert: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[31] E. Alsentzer, J. R. Murphy, W. Boag, W.-H. Weng, D. Jin, T. Naumann, and M. McDermott, "Publicly available clinical bert embeddings," *arXiv preprint arXiv:1904.03323*, 2019.

[32] K. Ameri, M. Hempel, H. Sharif, J. Lopez Jr, and K. Perumalla, "Cybert: Cybersecurity claim classification by fine-tuning the bert language model," *Journal of Cybersecurity and Privacy*, vol. 1, no. 4, pp. 615–637, 2021.

[33] S. S. Das, E. Serra, M. Halappanavar, A. Pothen, and E. Al-Shaer, "V2w-bert: A framework for effective hierarchical multiclass classification of software vulnerabilities," in *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–12, IEEE, 2021.

[34] Y. Chen, J. Ding, D. Li, and Z. Chen, "Joint bert model based cybersecurity named entity recognition," in *2021 The 4th International Conference on Software Engineering and Information Management*, pp. 236–242, 2021.

[35] S. Zhou, J. Liu, X. Zhong, and W. Zhao, "Named entity recognition using bert with whole world masking in cybersecurity domain," in *2021 IEEE 6th International Conference on Big Data Analytics (ICBDA)*, pp. 316–320, IEEE, 2021.

[36] C. Gao, X. Zhang, and H. Liu, "Data and knowledge-driven named entity recognition for cyber security," *Cybersecurity*, vol. 4, no. 1, pp. 1–13, 2021.

[37] J. Yin, M. Tang, J. Cao, and H. Wang, "Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description," *Knowledge-Based Systems*, vol. 210, p. 106529, 2020.

[38] A. Khazaei, M. Ghasemzadeh, and V. Derhami, "An automatic method for cvss score prediction using vulnerabilities description," *Journal of Intelligent & Fuzzy Systems*, vol. 30, no. 1, pp. 89–96, 2016.

[39] C. Elbaz, L. Rilling, and C. Morin, "Fighting n-day vulnerabilities with automated cvss vector prediction at disclosure," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pp. 1–10, 2020.

[40] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, pp. 29–48, Citeseer, 2003.

[41] "National vulnerability database," 2018.

[42] S. Neuhaus and T. Zimmermann, "Security trend analysis with cve topic models," in *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ISSRE '10, (Washington, DC, USA), pp. 111–120, IEEE Computer Society, 2010.

[43] S. Na, T. Kim, and H. Kim, "A study on the classification of common vulnerabilities and exposures using naïve bayes," in *Advances on Broad-Band Wireless Computing, Communication and Applications* (L. Barolli, F. Xhafa, and K. Yim, eds.), (Cham), pp. 657–662, Springer International Publishing, 2017.

[44] S. Rehman and K. Mustafa, "Software design level vulnerability classification model," *International Journal of Computer Science and Security (IJCSS)*, vol. 6, no. 4, pp. 235–255, 2012.

[45] M. Palmer, D. Gildea, and N. Xue, "Semantic role labeling," *Synthesis Lectures on Human Language Technologies*, vol. 3, no. 1, pp. 1–103, 2010.

[46] P. R. Kingsbury and M. Palmer, "From treebank to propbank.," in *LREC*, pp. 1989–1993, Citeseer, 2002.

[47] C.-M. Chen, J.-Y. Kan, Y.-H. Ou, Z.-X. Cai, A. Guan, *et al.*, "Threat action extraction using information retrieval," in *CS & IT Conference Proceedings*, vol. 11, CS & IT Conference Proceedings, 2021.

[48] H. Zhang, G. Shen, C. Guo, Y. Cui, and C. Jiang, "Ex-action: Automatically extracting threat actions from cyber threat intelligence report based on multimodal learning," *Security and Communication Networks*, vol. 2021, 2021.

[49] G. Ayoade, S. Chandra, L. Khan, K. Hamlen, and B. Thuraisingham, "Automated threat report classification over multi-source data," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pp. 236–245, IEEE, 2018.

[50] K. Satvat, R. Gjomemo, and V. Venkatakrishnan, "Extractor: extracting attack behavior from threat reports," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 598–615, IEEE, 2021.

[51] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank: An annotated corpus of semantic roles," *Computational linguistics*, vol. 31, no. 1, pp. 71–106, 2005.

[52] G. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: automated collection of vulnerabilities and their fixes from open-source software," in *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 30–39, 2021.

[53] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah, "Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence," CCS '16, (New York, NY, USA), p. 755â766, Association for Computing Machinery, 2016.

[54] R. Baeza-Yates, B. Ribeiro-Neto, *et al.*, *Modern information retrieval*, vol. 463. ACM press New York, 1999.

[55] B. Mitra, N. Craswell, *et al.*, *An introduction to neural information retrieval*. Now Foundations and Trends Boston, MA, 2018.

[56] T. Chen, Y. Sun, Y. Shi, and L. Hong, "On sampling strategies for neural network-based collaborative filtering," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 767–776, 2017.

APPENDIX A: Vulnerability Types Definitions

#	Vulnerability Type	Connected MITRE ATT&CK Techniques
1	General Improper Access Control	See the Functionality Section
2	Authentication Bypass by Capture-replay	**T1190** (Exploit Public-Facing Application) / **T1040** (Network Sniffing)
3	Improper Restriction of Excessive Authentication Attempts	**T1078** (Valid Accounts) / **T1110.001** (Brute Force: Password Guessing)
4	Overly Restrictive Account Lockout Mechanism	**T1446** (Device Lockout) / **T1531** (Account Access Removal) / **T1110** (Brute Force)
5	Use of Password Hash Instead of Password for Authentication	**T1550.002** (Use Alternate Authentication Material: Pass the Hash)
6	General Credential Management Errors	**T1552** (Unsecured Credentials) / **T1078** (Valid Accounts)
7	Cleartext Transmission of Sensitive Information	**T1552** (Unsecured Credentials) / **T1078** (Valid Accounts) / **T1040** (Network Sniffing)
8	Hard-coded Credentials	**T1078.001** (Default Accounts)
9	Weak Password/Hashing	**T1078** (Valid Accounts) / **T1110** (Brute Force)
10	General Cryptographic Issues	**T1078** (Valid Accounts) / **T1110** (Brute Force) / **T1557** (Man-in-the-Middle) / **T1040** (Network Sniffing)

11	XML External Entity (XXE)	**T1059** (Command and Scripting Interpreter) / **T1005** (Data from Local System) / **T1046** (Network Service Scanning)
12	XML Entity Expansion (XEE)	**T1499.004** (Endpoint Denial of Service: Application or System Exploitation)
14	URL Redirection to Untrusted Site ('Open Redirect')	**T1036** (Masquerading) / **T1566.002** (Phishing: Spearphishing Link)
15	Cross-site Scripting (XSS)	**T1059.007** (Command and Scripting Interpreter: JavaScript/JScript) / **T1557** (Man-in-the-Browser) / -Stored: **T1189** (Drive-by Compromise) -Others **T1204.001** (User Execution: Malicious Link)
16	OS Command Injection	**T1059** (Command and Scripting Interpreter) / **T1133** (External Remote Service)
17	SQL Injection	**T1059** (Command and Scripting Interpreter) / **T1005** (Data from Local System), **T1505.003** (Server Software Component: Web Shell), **T1136** (Create Account) / **T1190** (Exploit Public-Facing Application) / **T1565.001** (Data Manipulation)
18	Code Injection	**T1059** (Command and Scripting Interpreter)

19	Directory Traversal (Relative and Absolute)	See the Functionality Section (File Processing) / See the Functionality Section (File Processing) / **T1202** (Indirect Command Execution)
20	Symlink Attacks	See the Functionality Section (File Processing / See the Functionality Section (File Processing / **T1202** (Indirect Command Execution)
21	Untrusted/ Uncontrolled/ Unquoted Search Path	**T1574** (Hijack Execution Flow)
22	Unrestricted File Upload	**T1505.003** (Server Software Component: Web Shell) / **T1059** (Command and Scripting Interpreter)
23	Deserialization of Untrusted Data	**T1059** (Command and Scripting Interpreter)
24	Infinite Loop	**T1499.004** (Endpoint Denial of Service: Application or System Exploitation)
25	Cross-site Request Forgery (CSRF)	**T1068** (Exploitation for Privilege Escalation) / **T1204.001** (User Execution: Malicious Link)
26	Session Fixation	**T1563** (Remote Service Session Hijacking)
27	Uncontrolled Resource Consumption	**T1499** (Endpoint Denial of Service)
28	Server-Side Request Forgery (SSRF)	**T1090** (Proxy) / **T1135** (Network Discovery) / **T1005** (Data from Local System) / **T1133** (External Remote Service)

Table A.1: Vulnerability types mappings to MITRE ATT&CK techniques by MITRE guideline.

APPENDIX B: Functionality to MITRE ATT&CK Technique Mappings

#	Functionality	Connected MITRE ATTACK Techniques
1	Modify Configuration	**T1478** (Install Insecure or Malicious Configuration)
2	Create Account	**T1136** (Create Account) / **T1078** (Valid Accounts)
3	Disable protections	**T1562** (Impair Defenses)
4	Restart/Reboot	**T1529** (System Shutdown/Reboot)
5	Install App	**T1476** (Deliver Malicious App via Other Means)
6	Read from Memory	**T1005** (Data from Local System)
7	Obtain sensitive information: Credentials	**T1552** (Unsecured Credentials)
8	Obtain sensitive information: Other data	**T1005** (Data from Local System)
9	Password Reset	**T1098** (Account Manipulation)
10	Read files	**T1005** (Data from Local System) / **T1003.008** (OS Credential Dumping: /etc/passwd and /etc/shadow) / **T1552.001** (Unsecured Credentials: Credentials in Files)
11	Delete files	**T1485** (Data Destruction) / **T1499.004** (Endpoint Denial of Service: Application or System Exploitation)

12	Create/Upload file	**T1505.003** (Server Software Component: Web Shell) / **T1059** (Command and Scripting Interpreter)
13	Write to existing file	**T1565.001** (Data Manipulation) / **T1059** (Command and Scripting Interpreter) / **T1574** (Hijack Execution Flow) / **T1554** (Compromise Client Software Binary)
14	Change ownership or permissions	**T1222** (File and Directory Permissions Modification)
15	Memory Modification (Memory Buffer Errors, Pointer Issues, Type Errors, etc.)	**T1574** (Hijack Execution Flow), **T1499.004** (Endpoint Denial of Service: Application or System Exploitation)
16	Memory Read (Memory Buffer Errors, Pointer Issues, Type Errors, etc.)	**T1005** (Data from Local System) / **T1499.004** (Endpoint Denial of Service: Application or System Exploitation) / **T1211** (Exploitation for Defense Evasion) / **T1212** (Exploitation for Credential Access)

Table B.1: Functionalities' mapping to MITRE ATT&CK techniques by MITRE guideline.

APPENDIX C: Examples of Automated CVEs to Functionalities Classification

z	**Functionality** (f_z)	**Actions** (V_z)	**Object** (O_z)
1	Create Account	add, build, create, establish, generate	account, user account, new user, another user, arbitrary user, ftp user, administrative user, admin user, standard user, root user, admin user, new username, administrator user, administrative user, client
2	Create Or Upload File	add, build, create, dump, upload, generate, transfer, share, transmit	arbitrary posts, content, data, database, directory, drive, existing files, folder, information in the back-end database, insert, log data, log file content, crafted image, crafted photo, data, database, file
3	Delete File	delete, destruction,eliminate, erase, expunge, flush, purge, remove, uninstall, vanish, wipe	arbitrary posts, content, data, database, directories, directory, drive, existing files, files, folder, information in the back-end database, log data, log file

4	Disable Protections	abort, alter, block, corrupt, deactivate. destroy, disable, disconnect, disrupt, downgrade, evade, hinder, impair, interrupt, kill, modify, prevent, reduce, revoke, stop, shut down, terminate, turn off	anti spam, antivirus, antivirus, authentication, authorization, cryptographic protection mechanism, defense, dynamic malware analysis, firewall, guard, intrusion detection, ipsec, protection, secure file copy, security control, security update, shield, signature-based threat detection, ssh, ssl, code signing check, tls, tracking, VPN tunnel
5	Install App	deploy, deliver, install, setup	adware, app, application, crafted request, crafted web request, extension, malicious package, malicious web request, malware, package, phishing, phishing link, place, plugin, program, ransomware, software, spyware, surveillanceware, trojan, virus, widget

6	Modify Configuration	alter, change, compromise, configure, decrypt, edit, elevate, disable, forge, infect, manipulate, modify, poison, rename, replace, restrict, update	management system, administrative setting, configuration, configurator, preference, settings, system management, system property
7	Read From Memory	copy, load, read	memory, buffer, kernel, stack, pointer
8	Read Files	copy, load, observe, open, view, visit	data, database, file, message
9	Memory Read (Memory Buffer Errors, Pointer Issues, Type Errors, etc.)	copy, load, overread, underread, read	active memory, arbitrary kernel memory, arbitrary memory, buffer content, kernel, memory content, memory location, physical memory, process memory, restricted memory, sensitive memory, sensitive memory content, stack memory

10	Memory Modification (Memory Buffer Errors, Pointer Issues, Type Errors, etc.)	change, compromise, configure, forge, infect, manipulate, modify, overwrite, poison, replace, underwrite, update, write	active memory, arbitrary kernel memory, arbitrary memory, buffer content, kernel, memory content, memory location, physical memory, process memory, restricted memory, sensitive memory, sensitive memory content, stack memory
11	Obtain Sensitive Information - Credentials	access, acquire, capture, collect, crack, decrypt, disclose, discover, download, enumerate, expose, extract, find, gain, gather, get, guess, hijack, identify, locate, obtain, reach, retrieve, reveal, scrape, steal, traverse	/shadow, credential, key, /passwd, admin cookie, administrative login access, credentials, cryptographic, passcode, passcodes, password, passwords, plaintext credential, plaintext password, private key, sensitive credential information, session key, user accounts, usernames, user_login, user_pass, username

12	Obtain Sensitive Information - Other Data	access, acquire, capture, collect, disclose, discover, download, enumerate, expose, extract, find, gain, gather, get, guess, hijack, identify, locate, obtain, reach, retrieve, reveal, scrape, steal, traverse	configuration, cookie, database, information, sensitive, session id, string length, token value
13	Password Reset	change, compromise, configure, forge, infect, manipulate, modify, overwrite, poison, replace, update, write	etcpasswd, account, account information, admin, credential, e-mail, email, password, session_key, session key, user_name, user_pass, username
14	Change Ownership or Permissions	change, compromise, configure, decrypt, forge, infect, manipulate, modify, poison, replace, restrict, update	access control list, access to files, delete access, modify access, ownership, read-write permission, read access, read permission, read-write permission, read/write permission, user access, write access, write permission

15	Restart Or Reboot	crash, reboot, restart, shutdown	appliance, camera, computer, crash, device, laptop, modem, phone, process, router, server, service, system
16	Write To Existing File	modify, add, alter, append, change, compromise, edit, forge, insert, manipulate, override, overwrite, poison, re-write, replace, rewrite, save, store, underwrite, update, write	arbitrary code, arbitrary files, content, database, existing files, source code

Table C.1: List of verbs and objects extracted to represent functionality classes

Table C.2: List of objects extracted to represent causal links in functionality classes

z	**Functionality** (f_z)	**Object** (O'_z)
9	Memory Read (Memory Buffer Errors, Pointer Issues, Type Errors, etc.)	buffer over-read, buffer overread condition, denial of service (heap-based buffer over-read), denial of service (out-of-bounds array access), denial of service (out-of-bounds read and memory corruption), denial of service (out-of-bounds read), out-of-bound read, out-of-bounds access, read past the allocated buffer, reads outside of bounds of heap allocated data
10	Memory Modification (Memory Buffer Errors, Pointer Issues, Type Errors, etc.)	denial of service (out-of-bounds write), out-of-bounds write, overwrite buffers
15	Restart Or Reboot	denial of service (application crash or hang), denial of service (browser crash), denial of service (deadlock), denial of service (device outage), denial of service (device reboot), denial of service (device reload), denial of service (host os crash), denial of service (panic), denial of service (reboot), denial of service (reset), to restart unexpectedly, to reboot

CVE Description	
Functionality	**Predicted**
1	CVE-2020-5250: In PrestaShop before version 1.7.6.4, when a customer edits their address, they can freely change the id_address in the form, and thus steal someone else's address. It is the same with CustomerForm, you are able to change the id_customer and change all information of all accounts. The problem is patched in version 1.7.6.4.
Password Reset Modify Configuration	**Password Reset: 7.1** **Modify Configuration: 1.21**
2	CVE-2020-15170: apollo-adminservice before version 1.7.1 does not implement access controls. If users expose apollo-adminservice to internet(which is not recommended), there are potential security issues since apollo-adminservice is designed to work in intranet and it doesn't have access control built-in. Malicious hackers may access apollo-adminservice apis directly to access/edit the application's configurations. To fix the potential issue without upgrading, simply follow the advice that do not expose apollo-adminservice to internet.
Modify Configuration	Disable Protections: 8.32 **Modify Configuration: 5.18**

3	CVE-2020-5253: NetHack before version 3.6.0 allowed malicious use of escaping of characters in the configuration file (usually .nethackrc) which could be exploited. This bug is patched in NetHack 3.6.0.
Modify Configuration	Modify Configuration: 11.27
4	CVE-2020-5231: In Opencast before 7.6 and 8.1, users with the role ROLE_COURSE_ADMIN can use the user-utils endpoint to create new users not including the role ROLE_ADMIN. ROLE_COURSE_ADMIN is a non-standard role in Opencast which is referenced neither in the documentation nor in any code (except for tests) but only in the security configuration. From the name â implying an admin for a specific course â users would never expect that this role allows user creation. This issue is fixed in 7.6 and 8.1 which both ship a new default security configuration.
Create Account	**Create Account: 12.3**
5	CVE-2013-6129: The install/upgrade.php scripts in vBulletin 4.1 and 5 allow remote attackers to create administrative accounts via the customerid, htmldata[password], htmldata[confirmpassword], and htmldata[email] parameters, as exploited in the wild in October 2013.
Create Account	**Create Account: 13.84**
6	CVE-2015-4051: Beckhoff IPC Diagnostics before 1.8 does not properly restrict access to functions in /config, which allows remote attackers to cause a denial of service (reboot or shutdown), create arbitrary users, or possibly have unspecified other impact via a crafted request, as demonstrated by a beckhoff.com:service:cxconfig:1#Write SOAP action to /upnpisapi.
Restart Or Reboot Create Account	**Restart Or Reboot: 11.53** **Create Account: 5.49**
7	CVE-2019-3758: RSA Archer, versions prior to 6.6 P2 (6.6.0.2), contain an improper authentication vulnerability. The vulnerability allows sysadmins to create user accounts with insufficient credentials. Unauthenticated attackers could gain unauthorized access to the system using those accounts.
Create Account	**Create Account: 10.56**
8	CVE-2019-3798: Cloud Foundry Cloud Controller API Release, versions prior to 1.79.0, contains improper authentication when validating user permissions. A remote authenticated malicious user with the ability to create UAA clients and knowledge of the email of a victim in the foundation may escalate their privileges to that of the victim by creating a client with a name equal to the guid of their victim.

Create Account	**Create Account: 11.43**
9	CVE-2019-18581: Dell EMC Data Protection Advisor versions 6.3, 6.4, 6.5, 18.2 versions prior to patch 83, and 19.1 versions prior to patch 71 contain a server missing authorization vulnerability in the REST API. A remote authenticated malicious user with administrative privileges may potentially exploit this vulnerability to alter the applicationâs allowable list of OS commands. This may lead to arbitrary OS command execution as the regular user runs the DPA service on the affected system.
Disable protections	**Disable Protections: 7.47**
10	CVE-2018-17908: WebAccess Versions 8.3.2 and prior. During installation, the application installer disables user access control and does not re-enable it after the installation is complete. This could allow an attacker to run elevated arbitrary code.
Disable protections	Install App: 10.31 **Disable Protections: 2.81**
11	CVE-2018-17892: NUUO CMS all versions 3.1 and prior, The application implements a method of user account control that causes standard account security features to not be utilized as intended, which could allow user account compromise and may allow for remote code execution.
Disable protections	Obtain Sensitive Information: Credentials: 8.02 **Disable Protections: 1.82**
12	CVE-2018-15397: A vulnerability in the implementation of Traffic Flow Confidentiality (TFC) over IPsec functionality in Cisco Adaptive Security Appliance (ASA) Software and Cisco Firepower Threat Defense (FTD) Software could allow an unauthenticated, remote attacker to cause an affected device to restart unexpectedly, resulting in a denial of service (DoS) condition. The vulnerability is due to an error that may occur if the affected software renegotiates the encryption key for an IPsec tunnel when certain TFC traffic is in flight. An attacker could exploit this vulnerability by sending a malicious stream of TFC traffic through an established IPsec tunnel on an affected device. A successful exploit could allow the attacker to cause a daemon process on the affected device to crash, which could cause the device to crash and result in a DoS condition.
Restart/Reboot	**Restart Or Reboot: 14.91**

13	CVE-2018-15397: A vulnerability in the implementation of Traffic Flow Confidentiality (TFC) over IPsec functionality in Cisco Adaptive Security Appliance (ASA) Software and Cisco Firepower Threat Defense (FTD) Software could allow an unauthenticated, remote attacker to cause an affected device to restart unexpectedly, resulting in a denial of service (DoS) condition. The vulnerability is due to an error that may occur if the affected software renegotiates the encryption key for an IPsec tunnel when certain TFC traffic is in flight. An attacker could exploit this vulnerability by sending a malicious stream of TFC traffic through an established IPsec tunnel on an affected device. A successful exploit could allow the attacker to cause a daemon process on the affected device to crash, which could cause the device to crash and result in a DoS condition.
Restart/Reboot	**Restart Or Reboot: 15.05**
14	CVE-2019-1817: A vulnerability in the web proxy functionality of Cisco AsyncOS Software for Cisco Web Security Appliance could allow an unauthenticated, remote attacker to cause a denial of service (DoS) condition on an affected device. The vulnerability is due to improper validation of HTTP and HTTPS requests. An attacker could exploit this vulnerability by sending a malformed HTTP or HTTPS request to an affected device. An exploit could allow the attacker to cause a restart of the web proxy process, resulting in a temporary DoS condition.
Restart/Reboot	**Restart Or Reboot: 14.06**
15	CVE-2020-3312: A vulnerability in the application policy configuration of Cisco Firepower Threat Defense (FTD) Software could allow an unauthenticated, remote attacker to gain unauthorized read access to sensitive data on an affected device. The vulnerability is due to insufficient application identification. An attacker could exploit this vulnerability by sending crafted traffic to an affected device. A successful exploit could allow the attacker to gain unauthorized read access to sensitive data.
Obtain Sensitive Information: Other Data	**Obtain Sensitive Information: Other Data: 8.7588**
16	CVE-2020-3477: A vulnerability in the CLI parser of Cisco IOS Software and Cisco IOS XE Software could allow an authenticated, local attacker to access files from the flash: filesystem. The vulnerability is due to insufficient application of restrictions during the execution of a specific command. An attacker could exploit this vulnerability by using a specific command at the command line. A successful exploit could allow the attacker to obtain read-only access to files that are located on the flash: filesystem that otherwise might not have been accessible.

Read Files, Obtain Sensitive Information: Other Data	**Read Files: 9.47** **Obtain Sensitive Information: Other Data: 5.19**
17	CVE-2019-15963: A vulnerability in the web-based management interface of Cisco Unified Communications Manager could allow an authenticated, remote attacker to view sensitive information in the web-based management interface of the affected software. The vulnerability is due to insufficient protection of user-supplied input by the web-based management interface of the affected service. An attacker could exploit this vulnerability by accessing the interface and viewing restricted portions of the software configuration. A successful exploit could allow the attacker to gain access to sensitive information or conduct further attacks.
Obtain Sensitive Information: Other Data	**Obtain Sensitive Information: Other Data: 11.4**
18	CVE-2020-11045: In FreeRDP after 1.0 and before 2.0.0, there is an out-of-bound read in in update_read_bitmap_data that allows client memory to be read to an image buffer. The result displayed on screen as colour
Read From Memory	**Read From Memory: 7.54**
19	CVE-2018-7526: In TotalAlert Web Application in BeaconMedaes Scroll Medical Air Systems prior to v4107600010.23, by accessing a specific uniform resource locator (URL) on the webserver, a malicious user may be able to access information in the application without authenticating.
Obtain Sensitive Information: Other Data	**Obtain Sensitive Information: Other Data: 9.89**
20	CVE-2018-5445: A Path Traversal issue was discovered in Advantech WebAccess/SCADA versions prior to V8.2_20170817. An attacker has read access to files within the directory structure of the target device.
Read Files	**Read Files: 13.55**
21	CVE-2018-18990: LCDS Laquis SCADA prior to version 4.1.0.4150 allows a user-supplied path in file operations prior to proper validation. An attacker can leverage this vulnerability to disclose sensitive information under the context of the web server process.
Obtain Sensitive Information: Other Data	**Obtain Sensitive Information: Other Data: 10.04**
22	CVE-2020-16211: Advantech WebAccess HMI Designer, Versions 2.1.9.31 and prior. An out-of-bounds read vulnerability may be exploited by processing specially crafted project files, which may allow an attacker to read information.

Read Files Read From Memory		**Read Files: 7.61** **Read From Memory: 5.15**
23	CVE-2020-11652: An issue was discovered in SaltStack Salt before 2019.2.4 and 3000 before 3000.2. The salt-master process ClearFuncs class allows access to some methods that improperly sanitize paths. These methods allow arbitrary directory access to authenticated users.	
Obtain Sensitive Information: Other Data		Obtain Sensitive Information: Credentials: 6.79 **Obtain Sensitive Information: Other Data: 4.93**
24	CVE-2017-16651: Roundcube Webmail before 1.1.10, 1.2.x before 1.2.7, and 1.3.x before 1.3.3 allows unauthorized access to arbitrary files on the host's filesystem, including configuration files, as exploited in the wild in November 2017. The attacker must be able to authenticate at the target system with a valid username/password as the attack requires an active session. The issue is related to file-based attachment plugins and _task=settings_action=upload-display_from=timezone requests.	
Read Files		**Read Files: 12.42**
25	CVE-2019-5910: Directory traversal vulnerability in HOUSE GATE App for iOS 1.7.8 and earlier allows remote attackers to read arbitrary files via unspecified vectors.	
Read Files		**Read Files: 14.77**
26	CVE-2019-3787: Cloud Foundry UAA, versions prior to 73.0.0, falls back to appending âunknown.orgâ to a user's email address when one is not provided and the user name does not contain an @ character. This domain is held by a private company, which leads to attack vectors including password recovery emails sent to a potentially fraudulent address. This would allow the attacker to gain complete control of the user's account.	
Obtain sensitive information: Credentials		**Obtain Sensitive Information: Credentials: 7.69**
27	CVE-2019-3763: The RSA Identity Governance and Lifecycle software and RSA Via Lifecycle and Governance products prior to 7.1.0 P08 contain an information exposure vulnerability. The Office 365 user password may get logged in a plain text format in the Office 365 connector debug log file. An authenticated malicious local user with access to the debug logs may obtain the exposed password to use in further attacks.	
Obtain sensitive information: Credentials		**Obtain Sensitive Information: Credentials: 11.86**

28	CVE-2018-17900: Yokogawa STARDOM Controllers FCJ, FCN-100, FCN-RTU, FCN-500, All versions R4.10 and prior, The web application improperly protects credentials which could allow an attacker to obtain credentials for remote access to controllers.
Obtain sensitive information: Credentials	**Obtain Sensitive Information: Credentials: 12.61**
29	CVE-2019-6549: An attacker could retrieve plain-text credentials stored in a XML file on PR100088 Modbus gateway versions prior to Release R02 (or Software Version 1.1.13166) through FTP.
Obtain sensitive information: Credentials	**Obtain Sensitive Information: Credentials: 13.08**
30	CVE-2020-4408: The IBM QRadar Advisor 1.1 through 2.5.2 with Watson App for IBM QRadar SIEM does not adequately mask all passwords during input, which could be obtained by a physical attacker nearby. IBM X-Force ID: 179536.
Obtain sensitive information: Credentials	**Obtain Sensitive Information: Credentials: 8.46**
31	CVE-2019-13922: A vulnerability has been identified in SINEMA Remote Connect Server (All versions V2.0 SP1). An attacker with administrative privileges can obtain the hash of a connected device's password. The security vulnerability could be exploited by an attacker with network access to the SINEMA Remote Connect Server and administrative privileges. At the time of advisory publication no public exploitation of this security vulnerability was known.
Obtain sensitive information: Credentials	**Obtain Sensitive Information: Credentials: 12.54**
32	CVE-2018-7259: The FSX / P3Dv4 installer 2.0.1.231 for Flight Sim Labs A320-X sends a user's Google account credentials to http://installLog.flightsimlabs.com/LogHandler3.ashx if a pirated serial number has been entered, which allows remote attackers to obtain sensitive information, e.g., by sniffing the network for cleartext HTTP traffic. This behavior was removed in 2.0.1.232.
Obtain sensitive information: Credentials	**Obtain Sensitive Information: Credentials: 7.17**

33	CVE-2019-15956: A vulnerability in the web management interface of Cisco AsyncOS Software for Cisco Web Security Appliance (WSA) could allow an authenticated, remote attacker to perform an unauthorized system reset on an affected device. The vulnerability is due to improper authorization controls for a specific URL in the web management interface. An attacker could exploit this vulnerability by sending a crafted HTTP request to an affected device. A successful exploit could have a twofold impact: the attacker could either change the administrator password, gaining privileged access, or reset the network configuration details, causing a denial of service (DoS) condition. In both scenarios, manual intervention is required to restore normal operations.
Restart Or Reboot Password Reset	**Restart Or Reboot**: 11.48 Modify Configuration: 3.96 Delete Files: 2.32 Install App: 1.66 **Password Reset: 1.05**
34	CVE-2019-1915: A vulnerability in the web-based interface of Cisco Unified Communications Manager, Cisco Unified Communications Manager Session Management Edition (SME), Cisco Unified Communications Manager IM and Presence (Unified CM IMamp;P) Service, and Cisco Unity Connection could allow an unauthenticated, remote attacker to conduct a cross-site request forgery (CSRF) attack on an affected system. The vulnerability is due to insufficient CSRF protections by the affected software. An attacker could exploit this vulnerability by persuading a targeted user to click a malicious link. A successful exploit could allow the attacker to send arbitrary requests that could change the password of a targeted user. An attacker could then take unauthorized actions on behalf of the targeted user.
Password Reset	**Password Reset: 9.06**
35	CVE-2019-3775: Cloud Foundry UAA, versions prior to v70.0, allows a user to update their own email address. A remote authenticated user can impersonate a different user by changing their email address to that of a different user.
Password Reset	**Password Reset: 12.08**
36	CVE-2019-3782: Cloud Foundry CredHub CLI, versions prior to 2.2.1, inadvertently writes authentication credentials provided via environment variables to its persistent config file. A local authenticated malicious user with access to the CredHub CLI config file can use these credentials to retrieve and modify credentials stored in CredHub that are authorized to the targeted user.
Password Reset	**Password Reset: 8.88**

37	CVE-2019-3723: Dell EMC OpenManage Server Administrator (OMSA) versions prior to 9.1.0.3 and prior to 9.2.0.4 contain a web parameter tampering vulnerability. A remote unauthenticated attacker could potentially manipulate parameters of web requests to OMSA to create arbitrary files with empty content or delete the contents of any existing file, due to improper input parameter validation
Create Or Upload File Write To Existing File Delete Files	**Create Or Upload File: 9.58** **Write To Existing File: 5.9** **Delete Files: 5.49**
38	CVE-2019-3750: Dell Command Update versions prior to 3.1 contain an Arbitrary File Deletion Vulnerability. A local authenticated malicious user with low privileges potentially could exploit this vulnerability to delete arbitrary files by creating a symlink from the "Temp\IC\ICDebugLog.txt" to any targeted file. This issue occurs because of insecure handling of Temp directory permissions that were set incorrectly.
Delete files	**Delete Files: 13.84**
38	CVE-2020-1163: An elevation of privilege vulnerability exists in Windows Defender that leads arbitrary file deletion on the system.To exploit the vulnerability, an attacker would first have to log on to the system, aka 'Microsoft Windows Defender Elevation of Privilege Vulnerability'. This CVE ID is unique from CVE-2020-1170.
Delete files	**Delete Files: 12.92**
40	CVE-2020-15189: SOY CMS 3.0.2 and earlier is affected by Remote Code Execution (RCE) using Unrestricted File Upload. Cross-Site Scripting(XSS) vulnerability that was used in CVE-2020-15183 can be used to increase impact by redirecting the administrator to access a specially crafted page. This vulnerability is caused by insecure configuration in elFinder. This is fixed in version 3.0.2.328.
Create/Upload file	**Create Or Upload File: 13.11**
41	CVE-2020-5297: In OctoberCMS (october/october composer package) versions from 1.0.319 and before 1.0.466, an attacker can exploit this vulnerability to upload jpg, jpeg, bmp, png, webp, gif, ico, css, js, woff, woff2, svg, ttf, eot, json, md, less, sass, scss, xml files to any directory of an October CMS server. The vulnerability is only exploitable by an authenticated backend user with the `cms.manage_assets` permission. Issue has been patched in Build 466 (v1.0.466).
Create/Upload file	**Create Or Upload File: 15.2911**

42	CVE-2012-6081: Multiple unrestricted file upload vulnerabilities in the (1) twikidraw (action/twikidraw.py) and (2) anywikidraw (action/anywikidraw.py) actions in MoinMoin before 1.9.6 allow remote authenticated users with write permissions to execute arbitrary code by uploading a file with an executable extension, then accessing it via a direct request to the file in an unspecified directory, as exploited in the wild in July 2012.
Create/Upload file	**Create Or Upload File: 14.87**
43	CVE-2011-4106: TimThumb (timthumb.php) before 2.0 does not validate the entire source with the domain white list, which allows remote attackers to upload and execute arbitrary code via a URL containing a white-listed domain in the src parameter, then accessing it via a direct request to the file in the cache directory, as exploited in the wild in August 2011.
Create/Upload file	**Create Or Upload File: 12.71**
44	CVE-2016-3088: The Fileserver web application in Apache ActiveMQ 5.x before 5.14.0 allows remote attackers to upload and execute arbitrary files via an HTTP PUT followed by an HTTP MOVE request.
Create/Upload file	**Create Or Upload File: 14.73**
45	CVE-2020-3476: A vulnerability in the CLI implementation of a specific command of Cisco IOS XE Software could allow an authenticated, local attacker to overwrite arbitrary files in the underlying host file system. The vulnerability is due to insufficient validation of the parameters of a specific CLI command. An attacker could exploit this vulnerability by issuing that command with specific parameters. A successful exploit could allow the attacker to overwrite the content of any arbitrary file that resides on the underlying host file system.
Write to existing file	**Write To Existing File: 12.2**
46	CVE-2020-3440: A vulnerability in Cisco Webex Meetings Desktop App for Windows could allow an unauthenticated, remote attacker to overwrite arbitrary files on an end-user system. The vulnerability is due to improper validation of URL parameters that are sent from a website to the affected application. An attacker could exploit this vulnerability by persuading a user to follow a URL to a website that is designed to submit crafted input to the affected application. A successful exploit could allow the attacker to overwrite arbitrary files on the affected system, possibly corrupting or deleting critical system files.
Write to existing file	**Write To Existing File: 12.54**

47	CVE-2019-1836: A vulnerability in the system shell for Cisco Nexus 9000 Series Fabric Switches in Application Centric Infrastructure (ACI) mode could allow an authenticated, local attacker to use symbolic links to overwrite system files. These system files may be sensitive and should not be overwritable by non-root users. The attacker would need valid device credentials. The vulnerability is due to incorrect symbolic link verification of directory paths when they are used in the system shell. An attacker could exploit this vulnerability by authenticating to the device and providing crafted user input to specific symbolic link CLI commands. Successful exploitation could allow the attacker to overwrite system files that should be restricted. This vulnerability has been fixed in software version 14.1(1i).
Write to existing file	**Write To Existing File: 12.6**
48	CVE-2020-3237: A vulnerability in the Cisco Application Framework component of the Cisco IOx application environment could allow an authenticated, local attacker to overwrite arbitrary files in the virtual instance that is running on the affected device. The vulnerability is due to insufficient path restriction enforcement. An attacker could exploit this vulnerability by including a crafted file in an application package. An exploit could allow the attacker to overwrite files.
Write to existing file	**Write To Existing File: 12.79**
49	CVE-2008-4996: init in initramfs-tools 0.92f allows local users to overwrite arbitrary files via a symlink attack on the /tmp/initramfs.debug temporary file.
Write to existing file	**Write To Existing File: 11.4726**
50	CVE-2018-15392: A vulnerability in the DHCP service of Cisco Industrial Network Director could allow an unauthenticated, adjacent attacker to cause a denial of service (DoS) condition. The vulnerability is due to improper handling of DHCP lease requests. An attacker could exploit this vulnerability by sending malicious DHCP lease requests to an affected application. A successful exploit could allow the attacker to cause the DHCP service to terminate, resulting in a DoS condition.
Restart Or Reboot, Memory Read (Memory Errors)	**Restart Or Reboot: 13.34** **Memory Read (Memory Errors): 2.77**
51	CVE-2018-15392: A vulnerability in the DHCP service of Cisco Industrial Network Director could allow an unauthenticated, adjacent attacker to cause a denial of service (DoS) condition. The vulnerability is due to improper handling of DHCP lease requests. An attacker could exploit this vulnerability by sending malicious DHCP lease requests to an affected application. A successful exploit could allow the attacker to cause the DHCP service to terminate, resulting in a DoS condition.

Restart Or Reboot	**Restart Or Reboot: 13.34**
52	CVE-2020-5210: In NetHack before 3.6.5, an invalid argument to the -w command line option can cause a buffer overflow resulting in a crash or remote code execution/privilege escalation. This vulnerability affects systems that have NetHack installed suid/sgid and shared systems that allow users to influence command line options. Users should upgrade to NetHack 3.6.5.
Memory Read (Memory Errors)	**Memory Read (Memory Errors): 6.43**
53	CVE-2020-11019: In FreeRDP less than or equal to 2.0.0, when running with logger set to "WLOG_TRACE", a possible crash of application could occur due to a read of an invalid array index. Data could be printed as string to local terminal. This has been fixed in 2.1.0.
Memory Read (Memory Errors)	**Restart Or Reboot: 11.86** **Memory Read (Memory Errors): 2.35**
54	CVE-2020-15137: All versions of HoRNDIS are affected by an integer overflow in the RNDIS packet parsing routines. A malicious USB device can trigger disclosure of unrelated kernel memory to userspace applications on the host, or can cause the kernel to crash. Kernel memory disclosure is especially likely on 32-bit kernels; 64-bit kernels are more likely to crash on attempted exploitation. It is not believed that kernel memory corruption is possible, or that unattended kernel memory disclosure without the collaboration of a userspace program running on the host is possible. The vulnerability is in `HoRNDIS::receivePacket`. `msg_len`, `data_ofs`, and `data_len` can be controlled by an attached USB device, and a negative value of `data_ofs` can bypass the check for `(data_ofs + data_len + 8) msg_len`, and subsequently can cause a wild pointer copy in the `mbuf_copyback` call. The software is not maintained and no patches are planned. Users of multi-tenant systems with HoRNDIS installed should only connect trusted USB devices to their system.
Memory Read (Memory Errors)	**Memory Read (Memory Errors): 7.3**
55	CVE-2020-4068: In APNSwift 1.0.0, calling APNSwiftSigner.sign(digest:) is likely to result in a heap buffer overflow. This has been fixed in 1.0.1.
Memory Read (Memory Errors)	**Memory Read (Memory Errors): 9.26**
56	CVE-2020-15199: In Tensorflow before version 2.3.1, the `RaggedCountSparseOutput` does not validate that the input arguments form a valid ragged tensor. In particular, there is no validation that the `splits` tensor has the minimum required number of elements. Code uses this quantity to initialize a different data structure. Since `BatchedMap` is equivalent to a vector, it needs to have at least one element to not be `nullptr`. If user passes a `splits` tensor that is empty or has exactly one element, we get a `SIGABRT` signal raised by the operating system.

Memory Read (Memory Errors)	Restart Or Reboot: 4.26 **Memory Read (Memory Errors): 3.88**
57	CVE-2020-11068: In LoRaMac-node before 4.4.4, a reception buffer overflow can happen due to the received buffer size not being checked. This has been fixed in 4.4.4
Memory Read (Memory Errors)	**Memory Read (Memory Errors): 9.9**
58	CVE-2020-8649: There is a use-after-free vulnerability in the Linux kernel through 5.5.2 in the vgacon_invert_region function in drivers/video/console/vgacon.c.
Memory Read (Memory Errors)	**Memory Read (Memory Errors): 9.44**
59	CVE-2020-12652: The __mptctl_ioctl function in drivers/message/fusion/mptctl.c in the Linux kernel before 5.4.14 allows local users to hold an incorrect lock during the ioctl operation and trigger a race condition, i.e., a "double fetch" vulnerability, aka CID-28d76df18f0a. NOTE: the vendor states "The security impact of this bug is not as bad as it could have been because these operations are all privileged and root already has enormous destructive power."
Memory Read (Memory Errors)	**Memory Read (Memory Errors): 8.22**
60	CVE-2019-12660: A vulnerability in the CLI of Cisco IOS XE Software could allow an authenticated, local attacker to write values to the underlying memory of an affected device. The vulnerability is due to improper input validation and authorization of specific commands that a user can execute within the CLI. An attacker could exploit this vulnerability by authenticating to an affected device and issuing a specific set of commands. A successful exploit could allow the attacker to modify the configuration of the device to cause it to be non-secure and abnormally functioning.
Memory Modification (Memory Errors)	**Memory Modification (Memory Errors): 10.97**
61	CVE-2019-13522: An attacker could use a specially crafted project file to corrupt the memory and execute code under the privileges of the EZ PLC Editor Versions 1.8.41 and prior.
Memory Modification (Memory Errors)	**Memory Modification (Memory Errors): 11.15**
62	CVE-2018-10620: AVEVA InduSoft Web Studio v8.1 and v8.1SP1, and InTouch Machine Edition v2017 8.1 and v2017 8.1 SP1 a remote user could send a carefully crafted packet to exploit a stack-based buffer overflow vulnerability during tag, alarm, or event related actions such as read and write, with potential for code to be executed.

Memory Read (Memory Errors), Memory Modification (Memory Errors)	**Memory Read (Memory Errors): 4.84** **Memory Modification (Memory Errors): 1.84**
63	CVE-2019-12660: A vulnerability in the CLI of Cisco IOS XE Software could allow an authenticated, local attacker to write values to the underlying memory of an affected device. The vulnerability is due to improper input validation and authorization of specific commands that a user can execute within the CLI. An attacker could exploit this vulnerability by authenticating to an affected device and issuing a specific set of commands. A successful exploit could allow the attacker to modify the configuration of the device to cause it to be non-secure and abnormally functioning.
Memory Modification (Memory Errors), Disable Protections	**Memory Modification (Memory Errors): 10.97** **Disable Protections: 2.81**
64	CVE-2018-15376: A vulnerability in the embedded test subsystem of Cisco IOS Software for Cisco 800 Series Industrial Integrated Services Routers could allow an authenticated, local attacker to write arbitrary values to arbitrary locations in the memory space of an affected device. The vulnerability is due to the presence of certain test commands that were intended to be available only in internal development builds of the affected software. An attacker could exploit this vulnerability by using these commands on an affected device. A successful exploit could allow the attacker to write arbitrary values to arbitrary locations in the memory space of the affected device.
Memory Modification (Memory Errors)	**Memory Modification (Memory Errors): 11.77**
65	CVE-2019-1052: A remote code execution vulnerability exists in the way that the Chakra scripting engine handles objects in memory in Microsoft Edge, aka 'Chakra Scripting Engine Memory Corruption Vulnerability'.
Memory Read (Memory Errors)	**Memory Read (Memory Errors): 10.11**
66	CVE-2020-3309: A vulnerability in Cisco Firepower Device Manager (FDM) On-Box software could allow an authenticated, remote attacker to overwrite arbitrary files on the underlying operating system of an affected device. The vulnerability is due to improper input validation. An attacker could exploit this vulnerability by uploading a malicious file to an affected device. A successful exploit could allow the attacker to overwrite arbitrary files on as well as modify the underlying operating system of an affected device.
Write To Existing File	**Write To Existing File: 10.73**

Table C.3: The evaluation of classifying 66 CVE into one or more functionalities based on only the description without considering the second input. The table shows the top K prediction for each CVE description where K equals to the total number of predictions until all the correct classes are predicted. The correct predictions are depicted by bold font.