ABSTRACT

Title of dissertation: Knowledge for Cyber Threat Intelligence

Sudip Mittal, Doctor of Philosophy, 2019

Dissertation directed by: Professor Anupam Joshi
Department of Computer Science and Electrical Engineering

Keeping up with threat intelligence is a must for a security analyst today. There is a volume of information present in 'the wild' that affects an organization. We need to develop an artificial intelligence system that scours the intelligence sources, to keep the analyst updated about various threats that pose a risk to her organization. A security analyst who is better 'tapped in' can be more effective.

In this thesis, we present, *Cyber-All-Intel* an artificial intelligence system to aid a security analyst. It is a system for knowledge extraction, representation and analytics in an end-to-end pipeline grounded in the cybersecurity informatics domain. It uses multiple knowledge representations like, vector spaces and knowledge graphs in a 'VKG structure' to store incoming intelligence. The system also uses neural network models to pro-actively improve its knowledge. We have also created a query engine and an alert system that can be used by an analyst to find actionable cybersecurity insights.

Knowledge for Cyber Threat Intelligence

by

Sudip Mittal

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Dedication

To my parents.

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I'd like to thank my advisor, Professor Anupam Joshi for giving me an invaluable opportunity to work on challenging and extremely interesting projects over the past four years. He has always made himself available for help and advice and there has never been an occasion when I've knocked on his door and he hasn't given me time. It has been a pleasure to work with and learn from such an extraordinary individual.

I would also like to thank Prof. Tim Finin. Thanks are due to Professor Yelena Yesha, Dr. Karuna Joshi and Dr. Claudia Pearce for agreeing to serve on my thesis committee and for sparing their invaluable time reviewing the manuscript.

My colleagues at the Accelerated Cognitive Cybersecurity Lab & Ebiquity Lab have enriched my graduate life in many ways and deserve a special mention. My interaction with Varish Mulwad, Prajit Kumar Das, Sandeep Nair Narayanan, Nisha Pillai, Nilavra Pathak, Priyanka Ranade have been very fruitful.

I owe my deepest thanks to my family - my mother, father and sister who have always stood by me and guided me through my career, and have pulled me through against impossible odds at times. Words cannot express the gratitude I owe them.

It is impossible to remember all, and I apologize to those I've inadvertently left out.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the broad domain of security, analysts and policy makers need knowledge about the state of the world to make critical operational/tactical as well as strategic decisions. One such source of knowledge is threat intelligence. It plays a vital role in helping a security analyst mount a defense and is nearly always dependent on global events; for example, consider the following timeline:

- Microsoft publishes exploit details and issues patches on March 14, 2017 for a critical vulnerability which allows remote code execution if an attacker sends specially crafted messages to a Microsoft Server Message Block 1.0 (SMBv1) server [41].
- An exploit named ETERNALBLUE is leaked by the 'Shadow Brokers' hacker group on April 14, 2017 affecting various versions of the Microsoft Windows operating system [11]. It uses the same vulnerability mentioned above.
- ETERNALBLUE was used during the *WannaCry* ransomware attack on May 12, 2017 [81].
- It was also exploited to carry out the *NotPetya* cyberattack on June 27, 2017 [51].

When we consider this example from the security analyst point of view, she may not be aware of recent ‘intelligence’ available in ‘the wild’ and/or may be lazy/slow in updating her security configurations. What we need is an artificial intelligence based system that aids the security analyst. Such an AI should strive to keep the analyst updated and also issue timely alerts. The work in this paper, aims to prototype a system that can scour OSINT sources for such information and make them accessible to an analyst. The better “tapped in” the analyst is to the potential threat landscape, the better they are able to detect attacks.

In modern enterprises, security analysts monitor threats in a security operations center (SoC) by *watchstanding*, akin to a lookout on a ship watching the environs for danger. Screens typically show warnings and alerts from individual products and detectors that the enterprise has installed. Watchstanding permits a highly trained security analyst to look at all the disparate pieces of information, and see if they ‘click together’ to form some pattern which might indicate an attack.

The detection efficacy of a security analyst depends on her operational and strategic knowledge about current security landscape and the associated intelligence. This enables her to better interpret the data from the Security Information and Event Management (SIEM) systems in the SoC. Specifically, the analyst is aided by her background knowledge regarding the context of the system (e.g., what kinds of applications are installed on their system, what the systems normal behavior pattern is, what potential vulnerabilities might exist, what information might an adversary be after?), and the external world (e.g., “intelligence” about what new attacks that exist in the wild or are being discussed as possibilities, hacktivists discussing attacking a particular country or organization, etc.). Unfortunately, the knowledge about these security vulnerabilities and planned attacks is scattered on the dark web vulnerability markets, user/product forums, social media services, blogs, etc. However, even the best of SIEM systems today do not effectively reason on "intelligence" about the state of the cyberworld, such as an analyst might obtain by talking to peers or by looking, for instance, at dark web updates, security blogs, etc.

1.1 Thesis Statement

It is possible to automate threat intelligence acquisition by designing autonomous systems that scour disparate open sources, extract information, and represent it in way that can provide actionable information to security professionals.

1.1.0.1 The current state of art:

Consider a modern enterprise that has cyber infrastructure – either informational or cyber-physical – that it must protect. Today such a system has a host of point systems that act independently from one another, e.g., an intrusion detection/protection system that scans network traffic at the gateways, firewalls that regulate connections, application specific gateways that do deep packet inspection hard-coded with the semantics of the application being protected, host-based monitors like tripwire, malware scanners, identity management and authentication systems (sometimes with biometrics), and so on. Each of these systems defends against a particular vulnerability, and is often very effective against attacks that are known, or attacks that seek to cause a sudden and significant harm to the system.

In more sophisticated systems, the alerts and warnings from such individual components are aggregated and dashboarded in a security operations center, and perhaps even correlated in some simple manner. Network security analysts monitor these by *watchstanding*, akin to a lookout on a ship watching the environs for danger. Watch-standing permits a highly trained network security analyst to look at all the disparate pieces of information, and see if they "click together" to form some pattern which might indicate an attack. The analyst is aided by her background knowledge regarding the context of the system (e.g., what kinds of applications are installed on their system, what the systems normal behavior pattern is, what potential vulnerabilities might exist, what information might an adversary be after?) and and the external world (e.g., "intelligence" about what new attacks that exist in the wild or are being discussed as possibilities, hacktivists discussing attacking a particular country or organization, etc.). However, even the best of security information and event management (SIEM) systems do not bring in "intelligence" about the state of the cyber world, such as an analyst might obtain by looking, for instance, at security blogs, NVD/CVE updates, and parts of the dark web.

1.2 Problem Definition

Given an initial set of software and hardware, we will look for various threat and vulnerability intelligence, using textual sources like dark web vulnerability markets, social networks, blogs, etc. we will then convert this intelligence to a knowledge graph and augment this knowledge with entity vector embeddings in an accelerated environment. We will also then create various semantic agents that utilize this vectorized knowledge to provide intelligence alerts, improve the underling knowledge, identify similar attacks, etc.

There are three major parts to the problem that we are trying to solve. The first part entails finding relevant intelligence about various cybersecurity threats and vulnerabilities. The second one includes augmenting the existing knowledge infrastructure with vector embeddings. In the third one, we create various semantic agents that use the knowledge graph and vector embeddings to solve various research tasks.

1.3 Cybersecurity Informatics Systems

In this thesis we present, a cyber security informatics system - '*Cyber-All-Intel*'. We also discuss a precursor to the Cyber-All-Intel system, called *CyberTwitter*. Both these systems aim to augment a security analyst's work flow by gathering open source intelligence (OSINT), representing it in a format ideal for agents and applications to understand.

We will first introduce the CyberTwitter system followed by the Cyber-All-Intel. More details are available in Chapter 3 and 5.

1.3.1 CyberTwitter

In this system, we begin by collecting Twitter data. In the collected tweets we identify, tag and extract various real world conceptual entities related to cybersecurity vulnerabilities such as means of an attack, consequences of an attack, affected software, hardware, vendors, etc. using a Security Vulnerability Concept Extractor (SVCE) [35]. Concepts and entities extracted by SVCE are then linked to existing concepts and entities present in external, publicly available semantic knowledge bases, to further enrich our extracted data. In our system, this information is represented as a set of RDF triples in a semantic knowledge graph. We allow analysts to describe a system profile which captures information about installed software and / or hardware. We develop an *intelligence* ontology and use it along with SWRL rules to address time sensitive nature of cybersecurity events. CyberTwitter performs reasoning using this system profile, data in the knowledge base and varying time slices to generate the most relevant and important alerts for human review. Given, the sometimes, unreliable nature of information on Twitter [24] along with the possibility of different local security and organizational policies, we believe that it's best for a human analyst to be *'in loop'* with the system.

1.3.2 Cyber-All-Intel

The system takes as input cybersecurity related text data from various unstructured sources like Dark Web, blogs, social media, National Vulnerability Databases (NVDs), newspaper articles, etc. and represent the extracted knowledge in the '*VKG* *structure*'. We extract, represent and integrate the knowledge present in a variety of Open Source Intelligence (OSINT) web fora as entities, then use the resulting knowledge graph and embeddings to obtain actionable cybersecurity information for the analyst.

In order to better protect the product in development, it is necessary to create a repository of known vulnerabilities in these open source libraries and projects. Threat intelligence about some of these projects can be mined using *traditional* sources like NIST's National Vulnerability Database (NVD)[1], United States Computer Emergency Readiness Team (US-CERT)[2], etc. Other sources which are more *non-traditional* are, Twitter, Reddit, blogs, and news. Non-traditional sources are faster than the traditional ones. There is a significant gap between initial vulnerability announcement and NVD release [58]. Vulnerability threat intelligence appears first on non-traditional sources [57]. Mining non-traditional sources is becoming really important.

The Cyber-All-Intel system will also inform a developer about potential threats and vulnerabilities that the product in development might inherit as a result of linking to a vulnerable open source project or library. We mine threat intelligence from issues and bugs raised on web-based hosting service for version control like, GitHub [2], GitLab [3], bitbucket [1], etc. These platforms have been used by developers to host and collaborate on source code development [8]. We extract vulnerabilities raised on these platforms and represent them in a security knowledge graph.

[1]`https://nvd.nist.gov/`

[2]`https://www.us-cert.gov/`

The knowledge graph then becomes a store for various vulnerabilities and exposures present in various open source projects, products and libraries (see Chapter 5). This knowledge can then be queried by various developers helping them create products that are secure from the ground up. Pervasive software security entails cyber supply chain risk management, where the developers are made aware of various threats present in libraries they link with the development of their products. We also create another application that can track installed software on a client machine, and then use the above mentioned knowledge graph to *reason* alerts for the security analyst. These alerts warn the analyst if an installed software is linked to a vulnerable open source library or project. We built a proof of concept for this application that runs on a linux installation.

The multilingual nature of these non-traditional sources is a potential hindrance for cyber-defense professionals, as they might be limited by their knowledge of different languages. Despite this significant issue, the role of language in addressing cyber threats has been under explored. Multilingual understanding, adds to the many challenges security analysts continue to encounter. The security industry is heavily dependent upon the security analyst's ability in using specialized experience to reason over the disparate pieces of intelligence data available on the web, in order to discover potential threats and attacks. In Cyber-All-Intel, we use the cyber threat translation system developed by Ranade et al. [55], to gather relevant non-English threat intelligence data from sources such as Twitter. The data is then fed into the system, which converts the English representation of threats into a machine understandable format defined using our UCO Ontology [71] in OWL. We have developed a proof of concept for this system that takes in Russian threat intelligence and asserts them in Cyber-All-Intel. This helps cyber-defense systems gain intelligence about threats mentioned in the Russian text. The acquired intelligence is then fed into an AI-based cyber defense system that generates conclusions from a cumulation of aggregated threat intelligence data.

The system is built using the VKG structure – a hybrid structure that combines knowledge graph and embeddings in a vector space. The structure creates a new representation for relations and entities of interest. In the VKG structure, the knowledge graph includes explicit information about various entities and their relations to each other grounded in an ontology[3]. The vector embeddings, on the other hand, include implicit information found in context where these entities occur in a corpus. The base ontology is enhanced to have relations that describe the vector embeddings associated with terms in the ontology.

The Cyber-All-Intel system also pro-actively tries to improve the underlying cybersecurity knowledge. The vector part of the VKG structure is used to improve the knowledge graph part and vice versa. We utilize powerful deep neural networks to automatically update the underlying knowledge. Such an ability allows the system to be more accurate and best assist the security analyst in her tasks.

We have also included two applications in the Cyber-All-Intel system, namely, an alert recommender and a query processing engine that leverage the advantages provided by the VKG structure. The security analyst can ask the Cyber-All-Intel system to issue alerts based on an organization's '*system profile*'. She can dig deeper

[3]`https://www.w3.org/standards/semanticweb/ontology`

into an alert by asking complex queries like, '*Raise an alert if, a vulnerability similar to denial of service is listed in MySQL*' and get an answer from the system.

1.4 Contributions

Major contributions presented in this thesis include:

- Creating a knowledge graph based preliminary system called 'CyberTwitter' (Chapter 3). The system takes in cyber threat intelligence shared on Twitter and creates a cybersecurity knowledge graph for the same.
- Populating unstructured cybersecurity knowledge in Vectorized Knowledge Graphs (VKG) (Chapter 4) and creating the Cyber-All-Intel system (Chapter 5).
- Cybersecurity Knowledge Representation Improvement: We aim to use the vector space to improve the knowledge graph representation and the knowledge graph to improve the vector space representation (See Chapter 5).
- Actionable Cybersecurity Insights from Vectorized Knowledge Graphs: we create agents that fully utilize the advantages provided by VKGs (Chapter 4) for query processing (Chapter 5), generating alerts for threats (Chapter 5), and finding similar attacks.

1.5 Document Structure

The rest of the thesis is as follows – In Chapter 2, we discuss the related work and some background on knowledge representation. We discuss the CyberTwitter system in Chapter 3. The VKG Structure has been discussed in Chapter 4. We describe the Cyber-All-Intel system architecture and pipeline in Chapter 5. We discuss cybersecurity knowledge improvement in Chapter 5. The query processing and alert generation applications have been discussed in Chapter 5. We present our experimental setup and evaluation in Chapter 6 and 7, we conclude in Chapter 8.

Chapter 2

Background & Related Work

In this chapter we present some background and related work in the field of knowledge graphs, vector space models, text extraction, and knowledge representation in cybersecurity.

2.1 Knowledge Representation for Cybersecurity

Knowledge graphs have been used in cybersecurity to combine data and information from multiple sources. Undercofer et al. created an ontology by combining various taxonomies for intrusion detection [77]. Kandefer et al. [32] created a data repository of system vulnerabilities and with the help of a systems analyst, trained a system to identify and prevent system intrusion. Takahashi et al. [73, 72] built an ontology for cybersecurity information based on actual cybersecurity operations focused on cloud computing-based services. Rutkowski et al. [60] created a cybersecurity information exchange framework, known as CYBEX. The framework describes how cybersecurity information is exchanged between cybersecurity entities on a global scale and how implementation of this framework will eventually minimize the disparate availability of cybersecurity information. Another insightful work by Xie et al. [82] discusses uncertainty modeling for cyber security centered around near real-time security analysis such as intrusion response. In this paper the authors use Bayesian networks to model uncertainty in enhanced security analysis.

Syed et al. [71] created the Unified Cybersecurity Ontology (UCO) that supports information integration and cyber situational awareness in cybersecurity systems. The ontology incorporates and integrates heterogeneous data and knowledge schema from different cybersecurity systems and most commonly-used cybersecurity standards for information sharing and exchange such as STIX [10] and CYBEX [60]. The UCO ontology has also been mapped to a number of existing cybersecurity ontologies as well as concepts in the Linked Open Data cloud.

2.2 Open Source Intelligence (OSINT) sources

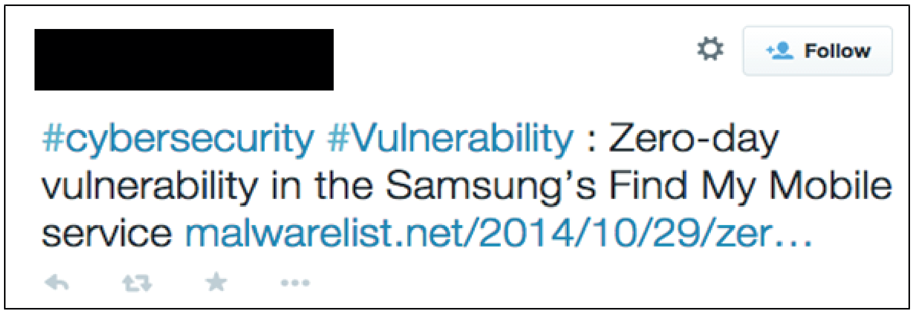
OSINT is intelligence gathered from publicly-available *overt* sources such as newspapers, magazines, social-networking sites, video sharing sites, wikis, blogs, etc. In cybersecurity domain, information available through OSINT can compliment data obtained through traditional security systems and monitoring tools like Intrusion Detection and Prevention Systems (IDPS) [46]. Cybersecurity information sources can be divided into two abstract groups, formal sources such as NIST's National Vulnerability Database (NVD), United States Computer Emergency Readiness Team (US-CERT), etc. and various informal sources such as blogs, developer forums, chat rooms and social media platforms like Twitter[1], Reddit[2] and Stackoverflow, these provide information related to security vulnerabilities, threats and attacks. A lot of information is published on these sources on a daily basis making it nearly impossible

[1]`https://twitter.com/hashtag/cybersecurity?lang=en`

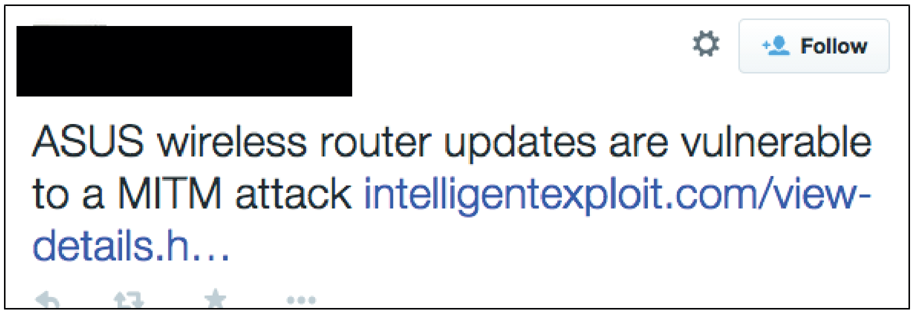
[2]`https://www.reddit.com/r/cybersecurity/`

for a human analyst to manually comb through, extract relevant information, and then understand various contextual scenarios in which an attack might take place. A manual approach even with a large number of human analysts would neither be efficient nor scalable. Automatically extracting relevant information from OSINT sources thus has received attention from the research community [48, 54, 38].

Over the past decade, Twitter has become a vital source for open source intelligence. The social media site's data has been used by researchers to gather intelligence about the impact of natural disasters [61] [19], terrorists attacks [52], government elections[74], predicting stock markets[14], etc. In our work, we are interested in using Twitter as a source of information to study various cybersecurity events. Twitter users as in when new vulnerabilities are made public, tweet about these vulnerabilities (Figure 2.1 and 2.2) to spread information on the network so that others can use that particular information to secure their systems. Individuals or reputed security experts like Brian Krebs (an investigative journalist who writes about cybercrime) can be valuable resources for cybersecurity incidents. Established companies like @web_security or @intersecww or disseminate news, tips and latest information on web security, web application protection, hacker incidents, data breaches, penetration testing results, etc. Other organization specific accounts like @githubstatus, @FirebaseStatus, @herokustatus, @stripestatus, @DOStatus (DigitalOcean), @redditstatus, @twitchstatus, @AdobeSecurity, @JuniperSIRT etc. report on security incidents with respect to their platforms and products. For obvious reasons such organizational accounts mentioned above are valuable sources of information with respect to cybersecurity events. We wish to use these Twitter updates


[redacted]

#cybersecurity #Vulnerability : Zero-day vulnerability in the Samsung's Find My Mobile service malwarelist.net/2014/10/29/zer...


[redacted]

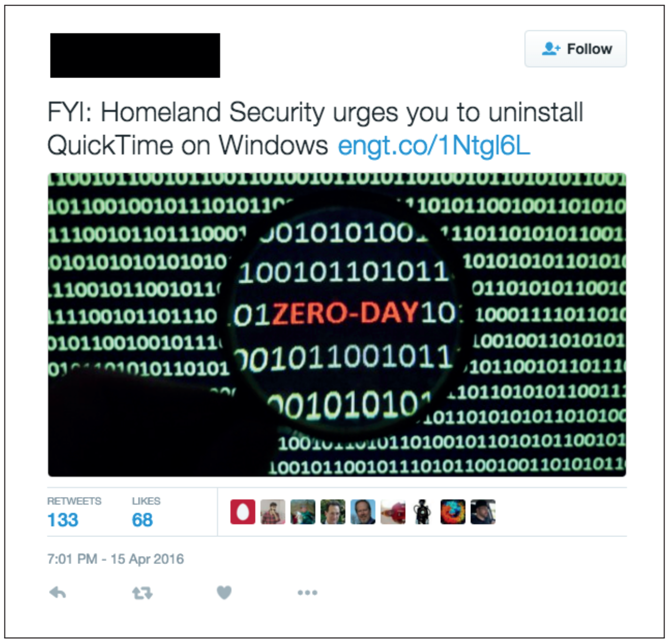
ASUS wireless router updates are vulnerable to a MITM attack intelligentexploit.com/view-details.h...

Figure 2.1: Both users and organizations use Twitter to report potential threats.

to mine intelligence about various cybersecurity threats and vulnerabilities, Chapter 3 gives details about our system.

2.3 Text Extraction for Cybersecurity

In our various preliminary systems [31, 45] we demonstrate the feasibility of automatically generating RDF linked data from vulnerability descriptions collected from the National Vulnerability Database [50], Twitter [76], etc. Joshi et al. [31] extract information on cybersecurity-related entities, concepts and relations which is then represented using custom ontologies for the cybersecurity domain and mapped to objects in the DBpedia knowledge base [9] using DBpedia Spotlight [40]. CyberTwitter [45], a framework to automatically issue cybersecurity vulnerability alerts to users. CyberTwitter converts vulnerability intelligence from tweets to RDF. It uses


[redacted]

Follow

FYI: Homeland Security urges you to uninstall QuickTime on Windows engt.co/1Ntgl6L

RETWEETS 133 LIKES 68

7:01 PM - 15 Apr 2016

Figure 2.2: Sample tweet from an individual user about a recent security vulnerability

the UCO ontology (Unified Cybersecurity Ontology) [71] to provide their system with cybersecurity domain information.

2.4 Vector Space Models & Knowledge Graphs

Extracting data from unstructured text (web) data sources, representing it, and reasoning over the representation to extract knowledge and information is one of the central challenges in the field of Artificial Intelligence. In addition to information extraction, it involves designing representations that capture the extracted information and that can be used to analyze it. There is an inherent information loss while representing knowledge through different methods. Consider two representations that are heavily used in literature – Knowledge Graphs and Vector Space

Embeddings. By representing knowledge as vector embeddings, we lose the explicit declarative character of the information. Knowledge graphs on the other hand are adept at asserting declarative information, but miss important contextual information around the entity or are restricted by the expressibility of the baseline ontology used to represent the knowledge [18].

It is important to highlight that both of the knowledge representation techniques provide applications built on these technologies certain advantages. Embeddings provide an easy way to search their neighborhood for similar concepts and can be used to create powerful deep learning systems for specific complex tasks. Knowledge graphs provide access to versatile reasoning techniques. Knowledge graphs also excel at creating rule-based systems where domain expertise can be leveraged. To overcome limitations of both and take advantage of their complementary strengths, *we propose the VKG structure that is part knowledge graph and part vector embeddings* (Chapter 4). VKG is more than the sum of these parts and can be used to develop powerful inference methods and a better semantic search.

Word embeddings are used to represent words in a continuous vector space. Two popular methods to generate these embeddings based on 'Relational Concurrence Assumption' are word2vec [42, 43] and GloVe [53]. The main idea behind generating embeddings for words is to say that vectors close together are semantically related. Word embeddings have been used in various applications like machine translation [69], improving local and global context [28], etc.

Modern knowledge graphs assert facts in the form of (*Subject*, *Predicate*, *Object*) triples, where *Subject* and *Object* are modeled as graph nodes and the edge between them (*Predicate*) model the relation between the two. DBpedia [9], YAGO (Yet Another Great Ontology) [68], YAGO2 [26], Google Knowledge Graph [64], etc. are some of the examples of popular knowledge graphs.

An important task on both vector space models and knowledge graphs is searching for similar entities, given an input entity. In vector spaces, embeddings close together are semantically related and various neighborhood search algorithms [23, 34] have been suggested. On the other hand semantic similarity on knowledge graphs using ontology matching, ontology alignment, schema matching, instance matching, similarity search, etc. remains a challenge [63, 20, 84]. In this paper we use the VKG structure, in which we link the knowledge graph nodes to their embeddings in a vector space (see Chapter 5).

Yang et al. [83] argued that a fast top-k search in knowledge graphs is challenging as both graph traversal and similarity search are expensive. The problem will get compounded as knowledge graphs increase in size. Their work proposes STAR, a top-k knowledge graph search framework to find top matches to a given input. Damljanovic et al. [17] have suggested using Random Indexing (RI) to generate a semantic index to an RDF graph [6]. These factors combined have led to an increased interest in semantic search, so as to access RDF data using Information Retrieval methods. We argue that vector embeddings can be used to search, as well as index entities in a knowledge graph. We have built a query engine on top of the VKG structure that removes the need to search on the knowledge graph and uses entity vector embeddings instead (see Chapter 4 and 7). However, queries that involve listing declarative knowledge and reasoning are done on the knowledge graph part of the VKG structure.

Vectorized knowledge graphs have also been created, systems like HOLE (holographic embeddings) [49] and TransE [80] learn compositional vector space representations of entire knowledge graphs by translating them to different hyperplanes. Our work is different from these models as we keep the knowledge graph part of the VKG structure as a traditional knowledge graph so as to fully utilize mature reasoning capabilities and incorporate the dynamic nature of the underlining corpus for our cybersecurity use-case. Vectorizing the entire knowledge graph part for a system like Cyber-All-Intel will have significant computational overhead because of the ever-changing nature of vulnerability relations and velocity of new input threat intelligence.

In another work thread different from ours, vector models have also been used for knowledge graph completion. Various authors have come up with models and intelligent systems to predict if certain nodes in the knowledge graphs should have a relation between them. The research task here is to complete a knowledge graph by finding out missing facts and using them to answer path queries [37, 47, 66, 25].

2.4.1 Open Source Software Security

Open source development has created a variety of new software security challenges. Closed source proponents claim that the availability of open source software's code allows hackers to easily find a way to compromise the security. They believe that "hackers finding the source code and placing back doors for unauthorized access to their systems is one of the biggest limiting factors for open source software" [36]. Closed source systems based on the principle of security through obscurity, may have theoretical or actual security vulnerabilities, but its owners or designers believe that they are more secure if the flaws are not known.

Security of open source systems stems from the Kerckhoffs's Law (sometimes refereed to as Shannon's maxim), which states that "A cryptosystem should be designed to be secure if everything is known about it except the key information." [62]. The strength behind open source software stems from it's wide audience looking at the code and collectively finding problems. In our work, we are interested in gathering threat intelligence about issue and vulnerability reports in open source repositories from project contributors.

2.4.2 Cybersecurity understanding across multiple languages

Cybersecurity terminology definitions differ across cultures and languages. The Department of Homeland Security started developing multilingual resources, to help link cybersecurity understanding across international governments [21]. Klavens et al. [33] outlines the importance of linguistics in the domain of security and claims language analysis propels understanding of communication between cyber-crime activist groups, filtering relevant data collection, and understanding the intention behind the words.

Chapter 3

CyberTwitter

3.1 CyberTwitter Framework

We develop CyberTwitter, a framework to automatically issue cybersecurity vulnerability alerts to users (Figure 3.1). CyberTwitter begins by collecting relevant tweets by querying the Twitter API. The tweet Collection module collects, cleans and stores tweets returned by the API. Every tweet is further processed by the Security Vulnerability Concept Extractor (SVCE) [35] which extracts various terms and concepts related to security vulnerabilities. Intelligence from these terms and concepts is then converted to RDF statements using our *intelligence* ontology. We use UCO ontology (Unified Cybersecurity Ontology) [71] to provide our system with cybersecurity domain information. RDF Linked Data representation is stored in our “Cybersecurity Knowledge Graph” allowing our alert system to reason over the data. Finally we issue alerts to the end user based on a “User System Profile”. We will further explain various details and sub-modules present in our system in the next few subsections.

Our system can be divided into two major parts. The first is a dynamically populated “Cybersecurity Knowledge Graph” that contains information about cybersecurity threats and vulnerabilities. The second is an alert system that issues alerts to the end user based on their “User System Profile” using the “Cybersecurity

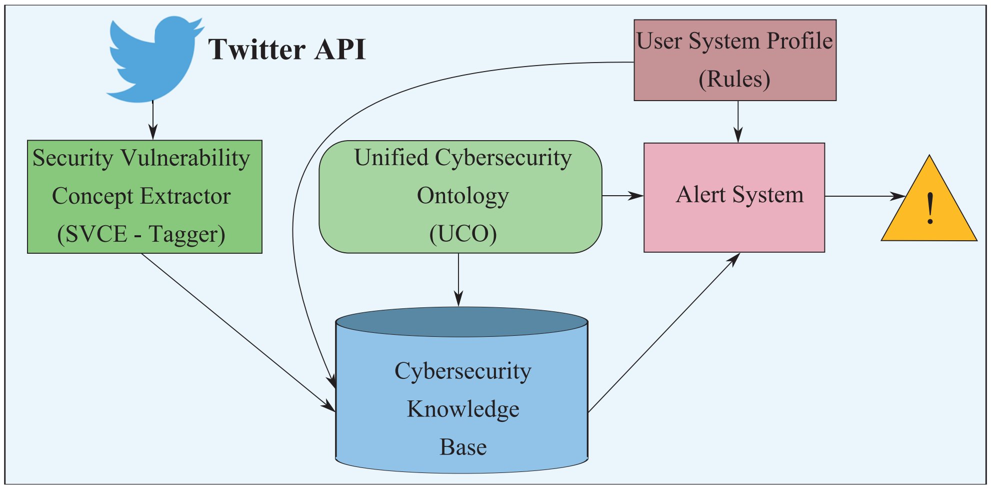

Twitter API
User System Profile (Rules)
Security Vulnerability Concept Extractor (SVCE - Tagger)
Unified Cybersecurity Ontology (UCO)
Alert System
!
Cybersecurity Knowledge Base

Figure 3.1: CyberTwitter: A framework for monitoring and analyzing tweets related to cyber attacks.

Knowledge Graph".

3.1.1 User System Profile

We obtain information about the user's system and store it in the "User System Profile" file. The profile contains information about the operating system, various installed softwares and their version information. We use the profile information as part of our rules. The system information is converted into SWRL rules [27] (see Section 3.1.6), that allows us to reason over them and generate cybersecurity alerts. A sample profile "User System Profile" is shown in Table 3.1.

Software	**Type**	**Version**
Ubuntu	Operating System	14.04
Adobe Flash	Software	11.2.202.616
Java	Software	7.0
Chromium Browser / Google Chrome	Browser	49.0.2623.112
Firefox	Browser	45.0.2
Adobe Flash Player (Chromium)	Extention	21.0.0.216-r1

Table 3.1: User System Profile.

3.1.2 Tweet Collection

CyberTwitter collects data through the Twitter Stream API[1] based on a set of keywords. These keywords are derived from the "User System Profile" and a list of cybersecurity terms (see Figure 3.2). For our system we limit ourselves to tweets in English language[2]. After collecting a good number of tweets we clean the data using WordNet, which is a large lexical database for English[44].

3.1.3 Security Vulnerability Concept Extractor

The Security Vulnerability Concept Extractor (SVCE) consists of a custom Named Entity Recognizer (NER) [35] which extracts terms related to security vulnerabilities. The NER was trained using text from security blogs, Common Vul-

[1]`https://dev.twitter.com/docs/streaming-api`

[2]`https://dev.twitter.com/streaming/overview/request-parameters#language`

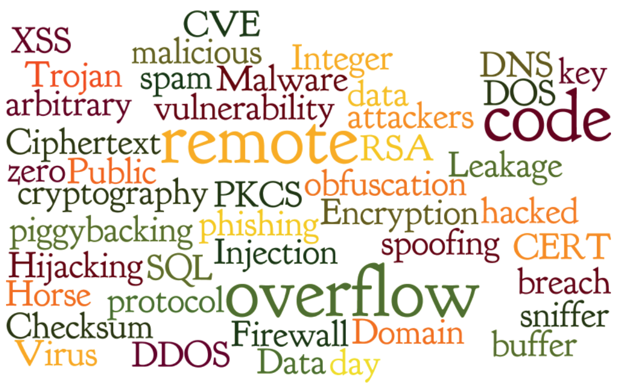


Figure 3.2: Data collection keywords.

nerabilities and Exposures (CVE) descriptions and official security bulletins from Microsoft and Adobe. It tags every sentence with the following concepts: Means of an attack, Consequence of an attack, affected software, hardware and operating system, version numbers, network related terms, file names and other technical terms.

The use of the custom NER provides us multiple advantages. SVCE discards all tweets for which the NER fails to identify even a single concept, thus further cleaning up the data. The extracted concepts are also used to generate an RDF Linked Data representation for every tweet that maybe queried by security systems to protect against potential attacks.

3.1.4 Filtering and Cleaning Data

In our "Cybersecurity Knowledge Graph" we wanted to store highly relevant tweets only. We filter tweets out based on the output of our Security Vulnerability Concept Extractor (SVCE). In our system we only keep those tweets which contain

```
Example tweet:

ASUS wireless router updates are vulnerable
to a MITM attack http://www.intelligent
exploit.com/view-details.html?id=20071

SVCE Output:

[[    (u'ASUS', u'PRODUCT,'),
        (u'wireless', u'OTHER,'),
        (u'router', u'OTHER,'),
        (u'updates', u'O'),
        (u'are', u'O'),
        (u'vulnerable', u'O'),
        (u'to', u'O'),
        (u'a', u'O'),
        (u'MITM', u'ATTACK,'),
        (u'attack', u'ATTACK,'),
        (u'http:www.intelligent
                exploit.comview-details.html?id=20071',
         u'O')]]
```

Figure 3.3: Labelled output generated by the Security Vulnerability Concept Extractor (SVCE).

two or more tags as generated by our SVCE. Such a threshold helps us realize the goal of including only highly relevant tweets in our knowledge graph.

3.1.5 Cybersecurity Ontologies and Knowledge Graphs

A data feed sent through the Twitter Stream API essentially consists of a stream of strings that computers can process. However, in the real world, strings represent terms and concepts that may sometimes be ambiguous and computers are not programmed to handle ambiguity. Computer systems can be aided in this task by various Semantic Web technologies that represent real world as concepts. These concepts are then associated with Uniform Resource Identifiers (URIs) [12]. For example, the string “Apple” can be associated with the company Apple Inc. or the fruit apple. Also, these concepts can have various attributes and relations to other concepts. An entity ‘Apple’ can have an attribute ‘type’ with a value ‘organization’ or ‘plant’. These attributes are vital so as to differentiate between two completely different concepts having same spellings.

For an intelligent system like CyberTwitter, it is vital to understand the difference between various real world concepts and also to posses a comprehensive knowledge about the cybersecurity domain. In this paper we use various publicly available cybersecurity ontologies and knowledge graphs to support information integration and cyber-situational awareness:

1. UCO: Unified Cybersecurity Ontology [71]: The ontology integrates heterogeneous data and knowledge schemas from different cybersecurity systems and

standards.

2. DBpedia[9]: DBpedia is a project to extract structured content from the information created as part of the Wikipedia project[3].

3. YAGO (Yet Another Great Ontology) [68]: It is a knowledge graph automatically extracted from Wikipedia and other sources.

We have used UCO to provide our system with knowledge about the cybersecurity domain. We use DBpedia and YAGO to link the output generated by our Security Vulnerability Concept Extractor (SVCE) to real world concepts. Entity matching process is performed by using DBpedia[4] [9] and YAGO[5] APIs with the MaxHits parameter set as 1. For example we can use DBpedia to map the string "Adobe Flash" to *dbr:Adobe_Flash* [6]. Both these external knowledge graphs help us map string entities to real world conceptual instances. The output from the SVCE module enlists various cybersecurity related entities in textual tweets like, Means of an attack, Consequence of an attack, affected software, etc. We use UCO, DBpedia and YAGO to link these entities to real world concepts. After entity linking we store the linked data as RDF triples [6] in our "Cybersecurity Knowledge Graph".

In our CyberTwitter system we need information of cybersecurity events. Events are temporal in nature. UCO though gives us a domain overview of cybersecurity it cannot handle temporal nature of events. So as to handle *time* in

[3]https://wikimediafoundation.org/wiki/Our_projects

[4]https://github.com/dbpedia-spotlight/dbpedia-spotlight

[5]https://github.com/yago-naga/aida

[6]http://dbpedia.org/page/Adobe_Flash

events we create an *Intelligence* ontology.

In our system we define 'Intelligence' as an actionable information for the human analyst which makes them aware about a new threat or vulnerability in a software / hardware that they list in their user system profile. The nature of intelligence in any security system is that it has a temporal dimension. A piece of information can be considered as vital information at a given time and useless at some other instance of time. So to incorporate time we included the following properties in the ontology:

1. *hasCounter(int:Intelligence, X)*: The number of tweets collected (X) with the given intelligence. This data property helps us attach a counter to the intelligence so as to map and group tweets with the intelligence they provide.

2. *hasBeginTime(int:Intelligence,, Y)*: This data property helps us mark the time when we got the first tweet (Y) that gives the system various details about a new vulnerability intelligence.

3. *hasLastIntelTime(int:Intelligence, Z)*: This data property helps us include the time stamp of the last tweet received (Z) with a particular intelligence.

4. *hasVulnerability(int:Intelligence, uco:Vulnerability)*: This object property holds an instance of the extracted vulnerability.

5. *productInUSP(int:Intelligence, L)*: This data property holds a boolean variable L which is set to 'True' if the vulnerability exists in one of the products listed in the 'user system profile'.

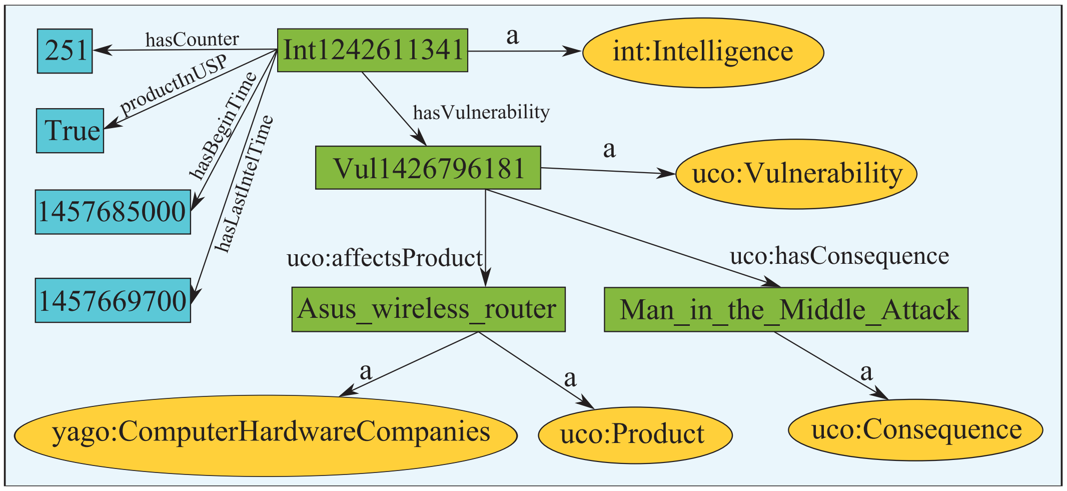


Figure 3.4: Graphical representation of RDF for example tweet shown in Figure 3.3.

6. *isCurrentlyValid(int:Intelligence, M)*: This runtime inferred data property holds a boolean value M which is set to 'True' if the intelligence entity is 'valid and current'. A valid and current intelligence is a one that gives details about an open, temporally significant vulnerability or threat in an affected software / hardware. This property is updated by various SWRL rules listed in Section 3.1.6.

To give an example Figure 3.4 shows a graphical representation of an intelligence, 'Int1242611341'. The particular intelligence instance is about a vulnerability 'Vul1426796181' that has a consequence of a 'man in the middle attack' that affects 'Asus_wireless_router'. The intelligence is supported by 251 number of tweets and the first tweet with this intelligence was received by the system at time 1457685000 and the latest tweet was received at time 1457669700. If the product is listed in the user system profile the boolean productInUSP data property is set to True.

Creating a comprehensive 'Cybersecurity Knowledge Graph' is vital for our system as it provides us with a set of rules and information in form of triples on which we can reason so as to issue vulnerability and threat alerts to the user. The end user can also be given access to the Knowledge Graph which they can query using a SPARQL interface [7] which is quite similar to SQL.

3.1.6 Alerts in CyberTwitter

In the final module of CyberTwitter we generate and issue alerts using the cybersecurity knowledge graph and the user system profile. After creating the knowledge graph we need an intelligent system to reason over various RDF statements and evaluate if the system should raise an alert to inform the user about a potential threat or vulnerability that may exist.

After creating the cybersecurity knowledge graph we include various SWRL rules [27] to our system. SWRL rules contain two parts, *antecedent* part (body), and a *consequent* (head). The body and head consist of conjunctions of a set of '*atoms*' [27]. Informally, a rule may be read as meaning that if the antecedent holds (is "true"), then the consequent must also hold.

We have logically divided this module in 2 different parts. In the first part we compute if an intelligence is 'valid and current' and in the second part we use a valid intelligence to raise an alert. When a tweet with actionable intelligence that already exists in the knowledge graph arrives in the system the intelligence entity corresponding to that vulnerability gets updated (For a tweet with new intelligence a new entity is created). When the alert system is triggered value of an 'inferred property' *isCurrentlyValid(int:Intelligence, M)* is computed through SWRL rules.

The first rule is used to compute the inferred property *isCurrentlyValid(int:Intelligence, M)* which depend on the value of last tweet time, if the product is in the user system profile and how 'old' is the intelligence. The variable T is a system parameter provided by the user so as to specify a time window. This time window determines if an intelligence entity is 'new' enough to issue an alert. For example, if the user sets T as 24 hours, then an intelligence entity which was last updated in the last 24 hour time period will be considered valid and current by the system.

```
hasLastIntelTime() ^
productInUSP() ^
withinRange( , CurrentSysTime - T ,CurrentSysTime)
=>
isCurrentlyValid()
```

The SWRL rules used to raise alerts use the inferred property *isCurrentlyValid(int:Intelligence, M)*, number of tweets associated with that intelligence entity. N is a system parameter specified by the user. This parameter can be used by the user to tweak the system so as to give alerts only if the number of tweets associated with an intelligence is substantial or if the system must inform the user about intelligence which are supported by a few tweets. For example, if the value of N set by the user is 10, then all intelligence entities with at-least 10 tweets supporting it are used to generate an alert.

```
Rule using consequences:
```

```
          isCurrentlyValid() ^
          hasConsequence() ^
          hasCounter() ^
          swrlb:greaterThanOrEqual(N, )
          =>
          RaiseAlert()


Rule using means:
          isCurrentlyValid() ^
          hasMeans() ^
          hasCounter() ^
          swrlb:greaterThanOrEqual(N, )
          =>
          RaiseAlert()
```

Using the above two rules we determine if various RDF statements have actionable intelligence that may be of interest to the user. and we issue alerts. In our system we have purposefully separated the two rules to generate the value for hasIntelligence. One for hasConsequence and another one for hasMeans. This can create multiple repeated alerts in our system. We can combine the two rules to produce a more concrete rule where both consequences and means are present in the RDF statement. However to ensure a better throughput and performance we use two different rules in our system.

After generating the alerts we display them to the human analyst along with a link to the list tweets and SVCE tags through which the particular alert was generated. These alerts can then be used by human analysts and policy makers to make vital decisions to secure their organizational / personal systems.

Chapter 4

VKG Structure

In this chapter, we describe our VKG structure, which leverages both vector spaces and knowledge graphs to create a new representation for relations and entities of interest present in text. In the VKG structure, an entity is represented as a node in a knowledge graph and is linked to its representation in a vector space. Figure 4.1 gives an example of the VKG structure where entity nodes are linked to each other using explicit relations as in a knowledge graph and are also linked to their word embeddings in a vector space. The VKG structure enables an application to reason over the knowledge graph portion of the structure and also run computations on the vector space part.

The VKG structure enables us to specifically assert information present in the vector representation of concepts and entities using semantic relations, for example, in Figure 4.1; using the VKG structure we can explicitly assert that the vector representation of 'Milo' and 'Cat' are related like, $< Milo, isA, Cat >$. 'Tom' and 'Milo' are related with the relation, $< Tom, hasPet, Milo >$. The vector representation of 'Milo' can be used to find other pets that are similar to it, but the explicit information that 'Milo' is a 'Cat' and it's 'Tom' that has a pet named 'Milo' can be accurately derived explicitly from the knowledge graph part.

The VKG structure helps us unify knowledge graphs and vector representation

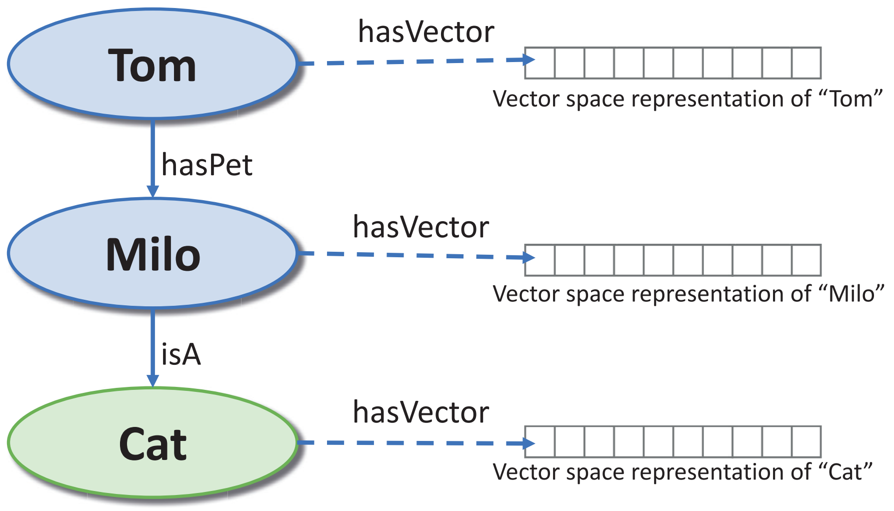

Tom
hasVector
Vector space representation of “Tom”
hasPet
Milo
hasVector
Vector space representation of “Milo”
isA
Cat
hasVector
Vector space representation of “Cat”

Figure 4.1: An example of a VKG structure representing “Tom has a pet Milo and Milo is a cat”.

of entities, and allows us to develop powerful inference methods that combine their complementary strengths.

The vector representation we use is based on the ‘Relational Concurrence Assumption’ highlighted in [42, 43, 16]. Word embeddings are able to capture different degrees of similarity between words. Mikolov et al. [42, 43] argue that embeddings can reproduce semantic and syntactic patterns using vector arithmetic. Patterns such as “Man is to Woman as King is to Queen” can be generated through algebraic operations on the vector representations of these words such that the vector representation of “King” - “Man” + “Woman” produces a result which is closest to the vector representation of “Queen” in the model. Such relationships can be generated for a range of semantic relations as well as syntactic relations. However, in spite of the fact that vector space models excel at determining similarity between two vectors they are severely constrained while creating complex dependency relations and other logic based operations that are a forte of various semantic web based applications [18, 13].

Knowledge graphs, on the other hand, are able to use powerful reasoning techniques to infer and materialize facts as explicit knowledge. Those based on description logic representation frameworks like OWL, for example, can exploit axioms implicit in the graphs to compute logical relations like consistency, concept satisfiability, disjointness, and subsumption. As a result, they are generally much slower while handling operations like, ontology alignment, instance matching, and semantic search [63, 29].

The intuition behind our approach is that an entity's context from its immediate neighborhood, present as word embeddings, adds more information along with various relations present explicitly in a knowledge graph. Entity representation in vector space gives us information present in the immediate context of the place where they occur in the text and knowledge graphs give us explicit information that may or may not be present in the specific piece of text. Embeddings can help find similar nodes or words faster using neighborhood search algorithms and search space reductions. They also support partial matching techniques. Knowledge graphs provide many reasoning tools including query languages like SPARQL[1], rule languages like SWRL[2], and description logic reasoners.

Potential applications that will work on our VKG structure, need to be de-

[1]`https://www.w3.org/TR/rdf-sparql-query/`

[2]`https://www.w3.org/Submission/SWRL/`

signed to take advantages provided by integrating vector space models with a knowledge graph. In a general efficient use-case for our VKG structure, 'fast' top-k search should be done on the vector space part aided by the knowledge graph, and the 'slow' reasoning based computations should be performed on just the knowledge graph part. An input query can be decomposed into sub-queries which run on respective parts of the VKG structure (see 5.4.1.1). We analogize this to thinking 'fast' in vector space along with thinking 'deeply' and 'slowly' by using the knowledge graph.

By using the VKG structure, we link entity vector embeddings with their knowledge graph nodes. Domain specific knowledge graphs are built using a schema that is generally curated by domain experts. When we link the nodes and embeddings we can use the explicit information present in these ontologies to provide domain understanding to embeddings in vector space. Adding domain knowledge to vector embeddings can further improve various applications built upon the structure. The vector embeddings can be used to train machine learning models for various tasks. These machine learning models can use explicit assertions present in the knowledge graph part. This will also help in improving the quality of the results generated for various input queries discussed in Chapter 5.

In the example, (Figure 4.1) we can use the VKG structure to state that the vector embedding of 'Milo' belongs to the class 'Cat', which is a subclass of 'Mammals' and so on. In a cybersecurity example (Figure 4.2) from our Cyber-All-Intel system discussed in Chapter 5, we explicitly state that 'denial_of_service' is a vulnerability. Using the VKG structure we connect the fact that the vector

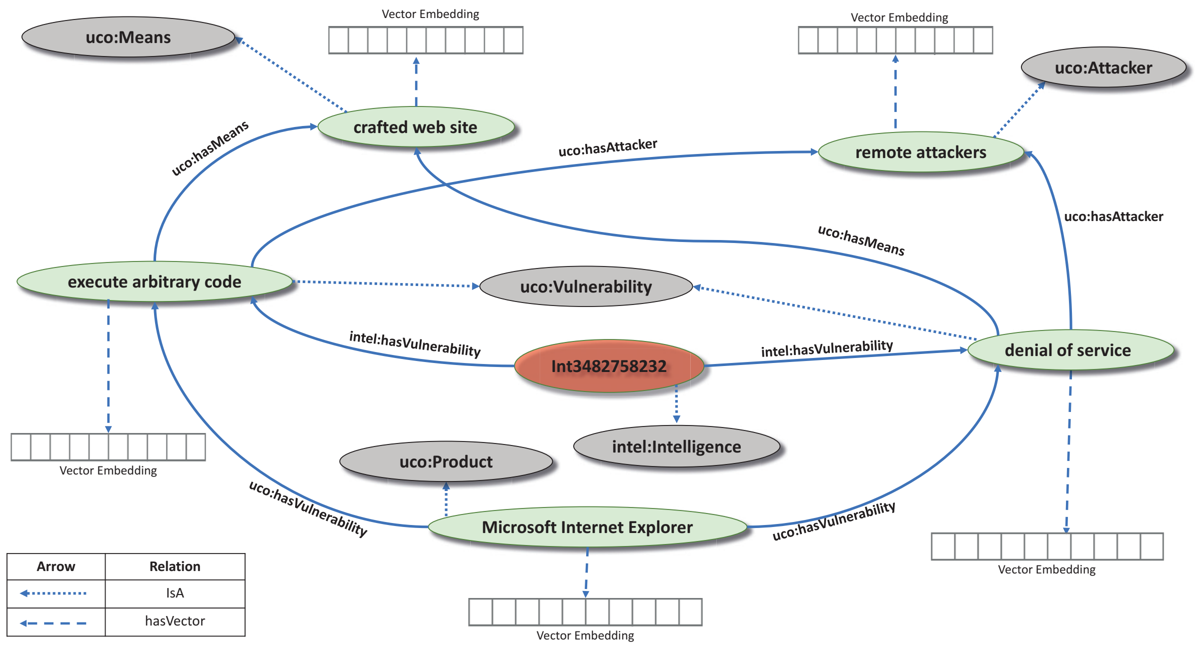


Figure 4.2: In the VKG structure for *"Microsoft Internet Explorer allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via a crafted web site, aka "Internet Explorer Memory Corruption Vulnerability."* the knowledge graph part asserted using UCO includes the information that a product 'Microsoft Internet Explorer' has vulnerabilities 'execute arbitrary code' and 'denial of service' that can be exploited by 'remote attackers' using the means 'crafted web site'. The knowledge graph entities are linked to their vector embeddings using the relation 'hasVector'.

embedding of 'denial_of_service' is a vulnerability.

Knowledge graph nodes in the VKG structure can be used to add information from other sources like DBpedia, YAGO, and Freebase. This helps integrate information that is not present in the input corpus. For example, in Figure 4.2 we can link using the '*owl* : *sameAs*' property 'Microsoft_Internet_Explorer' to its DBpedia equivalent 'dbp:Internet_Explorer'. Asserting this relation adds information like Internet Explorer is a product from Microsoft. This information may not have been present in the input cybersecurity corpus but is present in DBpedia.

4.1 Populating the VKG Structure

In order to create the VKG structure, the structure population system requires as input, a text corpus. The aim of the system is to create the VKG structure for the concepts and entities present in the input corpus which requires us to create the knowledge graph and the vector parts separately and then linking the two.

Various steps and technologies required are enumerated below. The Cyber-All-Intel system that uses the VKG structure to represent a textual corpus is described in Chapter **??**.

1. *Training vectors*: For the vector part of the VKG structure, we can generate entity embeddings using any of a number of vectorization algorithms. For text, many of these are based on the 'Relational Concurrence Assumption' principle [42, 43, 53, 28].

2. *Creating semantic triples*: An information extraction pipeline extracts a knowl-

edge graph from a collection of text documents. The first step applies components from Stanford CoreNLP components [39] trained to recognize entities and relations in the cybersecurity domain to produce a knowledge graph for each document. The second step uses components from Kelvin [22] to integrate the document-level graphs by performing cross-document co-reference, linking entities to reference knowledge bases like DBpedia, and drawing additional inferences. The resulting knowledge graph is then materialized as an RDF graph.

3. *Creating links between entity vectors and nodes*: We link knowledge graph nodes to their corresponding words in the vector space vocabulary using the $hasVector$ relationship as shown in Figure 4.1. Keeping the lexical tokens in the knowledge graph allows us to update vector embeddings as the underlining corpus changes and renders the vector model stale. This symbolic linking of the knowledge graph part and the vector part via the $hasVector$ relation is initiated after the RDF triples are generated.

4.2 Advantages

The VKG structure helps us unify knowledge graphs and vector representation of entities, and allows us to develop powerful inference methods that combine their complementary strengths.

The vector representation we use, enable us to encode 'local contextual knowledge'. These are based on the 'Relational Concurrence Assumption' highlighted in [42, 43, 16]. Word embeddings are able to capture different degrees of similarity between words. However, they are severely constrained while creating complex dependency relations and other logic based operations that are a forte of various semantic web based applications [18, 13].

Knowledge graphs, on the other hand, are able to use powerful reasoning techniques to infer and materialize facts as explicit 'global knowledge'. Those based on description logic representation frameworks like OWL, for example, can exploit axioms implicit in the graphs to compute logical relations like consistency, concept satisfiability, disjointness, and subsumption. As a result, they are generally much slower while handling operations like, ontology alignment, instance matching, and semantic search [63, 29]. Knowledge graphs provide many reasoning tools including query languages like SPARQL [7], rule languages like SWRL [27], and description logic reasoners.

Potential applications that will work on our VKG structure, need to be designed to take advantages provided by integrating vector space models with a knowledge graph. In a general efficient use-case for our VKG structure, 'fast' top-k search should be done on the vector space part aided by the knowledge graph, and the 'slow' reasoning based computations should be performed on just the knowledge graph part. An input query can be decomposed into sub-queries which run on respective parts of the VKG structure (see 5.4.1.1).

Domain specific knowledge graphs are built using a schema that is generally curated by domain experts. When we link the nodes and embeddings we can use the explicit information present in these ontologies to provide domain understanding to embeddings in vector space. Adding domain knowledge to vector embeddings can further improve various applications built upon the structure. The vector embeddings can be used to train machine learning models for various tasks. These machine learning models can use explicit assertions present in the knowledge graph part. This will also help in improving the quality of the results generated for various input queries discussed in Chapter 5.

Knowledge graph nodes in the VKG structure can be used to add information from other sources like, DBpedia, YAGO, and Freebase. This helps integrate information that is not present in the input corpus. For example, in Figure 4.2 we can link using the '$owl : sameAs$' property, 'Microsoft_Internet_Explorer' to its DBpedia equivalent 'dbp:Internet_Explorer' [5]. Asserting this relation adds information like Internet Explorer is a product from Microsoft. This information may not have been present in the input cybersecurity corpus but is present in DBpedia.

Another advantage provided by integrating vector spaces and knowledge graphs is that we can use both of them to improve the results provided by either of the parts alone. For example (in Figure 4.2), we can use the explicit information provided in the knowledge graph to aid the similarity search in vector space. If we are searching the vector space for entities similar to 'denial_of_service', we can further improve our results by ensuring the entities returned belong to class 'Vulnerability'. This information is available from the knowledge graph. This technique of knowledge graph aided vector space similarity search (*VKG Search*, See Chapter 5) is used in our query engine. We execute similarity search on the embeddings and then filter out entities using the knowledge graph.

In Chapter 5, we discuss applications we have created for the Cyber-All-Intel in detail.

Chapter 5

Cyber-All-Intel

In this chapter we discuss the overarching design and system architecture for Cyber-All-Intel (Figure 5.6). We first discuss various intelligence sources that serve as input to Cyber-All-Intel; then we go through the architecture and the rationale behind our design decisions. Later we explain the knowledge representation techniques (VKG structure) used in our system along with its advantages. We also discuss a few agents that can leverage these representation techniques to provide value to a security analyst.

5.1 Cybersecurity Sources

OSINT is intelligence gathered from publicly-available *overt* sources such as newspapers, magazines, social-networking sites, video sharing sites, wikis, blogs, etc. In cybersecurity domain, information available through OSINT can compliment data obtained through traditional security systems and monitoring tools like Intrusion Detection and Prevention Systems (IDPS) [46]. Cybersecurity information sources can be divided into two abstract groups, formal sources such as NIST's National Vulnerability Database (NVD), United States Computer Emergency Readiness Team (US-CERT), etc. and various informal sources such as blogs, developer forums, chat rooms and social media platforms like Twitter, Reddit [56] and Stackoverflow [67], these provide information related to security vulnerabilities, threats and attacks. A lot of information is published on these sources on a daily basis making it nearly impossible for a human analyst to manually comb through, extract relevant information, and then understand various contextual scenarios in which an attack might take place. A manual approach even with a large number of human analysts would neither be efficient nor scalable. Automatically extracting relevant information from OSINT sources thus has received attention from the research community [48, 54, 38].

5.1.1 Multilingual Sources

In Cyber-All-Intel, we also include multilingual open source intelligence sources. As a proof of concept, we include Russian threat intelligence data from Twitter. The data is then assimilated into a vector representation in order to bring semantically similar terms together [43]. The data is then fed into Cyber-All-Intel, which converts the English representation of the Russian data into a machine understandable format defined using our UCO Ontology [71] in OWL. This helps cyber-defense systems gain intelligence about threats mentioned in the Russian text. The acquired intelligence is then fed into an AI-based cyber defense system that generates conclusions from a cumulation of aggregated threat intelligence data.

The intelligence translation system that we discuss by Ranade et al. [55] helps us automate this process by taking data from a variety of multilingual sources, extracting, representing and integrating the knowledge present in it as embeddings and knowledge graphs, and then use the resulting artificial intelligence systems to provide actionable insights to SoC professionals. Figure 5.1 showcases our pipeline, which takes in Russian threat intelligence and stores it in as a *VKG structure*.

This data can augment both CyberTwitter and Cyber-All-Intel. The systems store threat intelligence in a knowledge representation that can be used by AI based cyber-defense systems (See Figure 5.1). Such systems generally have a knowledge representation engine, a reasoning engine, and few applications like an alert generation system, recommender system, query processing system, etc.

The knowledge representation system, converts input threat intelligence (usually in a textual format) into a machine readable format. In our system we represent it in RDF[1], with cybersecurity domain knowledge provided by the Unified Cybersecurity Ontology (UCO) [71]. The intelligence ontology [45] provides information about the intelligence domain. We also include specific conceptual embeddings for security concepts in our threat representation format. The knowledge reasoning part of the system provides domain specific reasoning capability generally encoded as logical rules by a domain expert. The applications and the reasoning engine generally use the machine readable representation to provide specific functionality. Figure 5.1 also provides the graph structure for the translated English intelligence: "*URL Command Injection Remote Code Execution Vulnerability in Microsoft Skype*". Figure 5.2 provides the RDF representation for the same intelligence.

[1]`https://www.w3.org/RDF/`

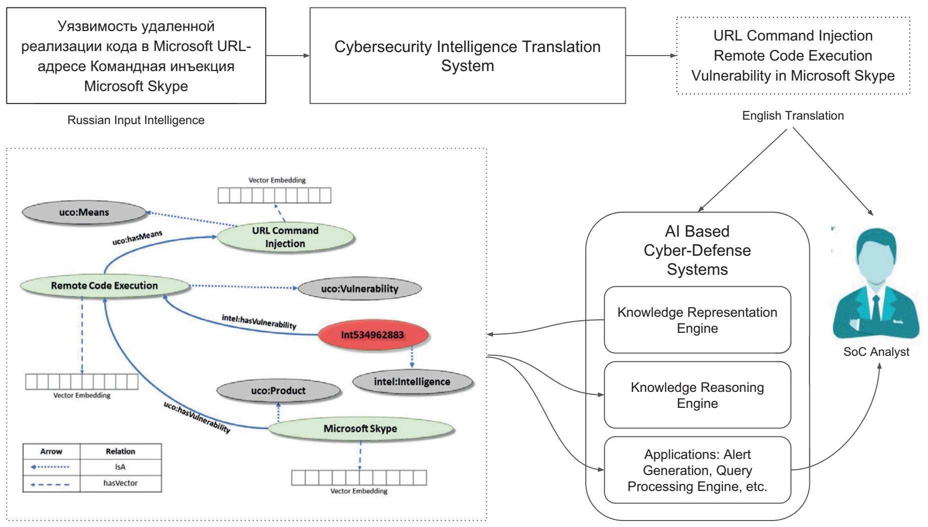

Уязвимость удаленной реализации кода в Microsoft URL-адресе Командная инъекция Microsoft Skype
Russian Input Intelligence
Cybersecurity Intelligence Translation System
URL Command Injection Remote Code Execution Vulnerability in Microsoft Skype
English Translation
Vector Embedding
uco:Means
uco:hasMeans
URL Command Injection
Remote Code Execution
uco:Vulnerability
intel:hasVulnerability
Int534962883
intel:Intelligence
uco:Product
uco:hasVulnerability
Microsoft Skype
Vector Embedding
Vector Embedding
Arrow
Relation
IsA
hasVector
AI Based Cyber-Defense Systems
Knowledge Representation Engine
Knowledge Reasoning Engine
Applications: Alert Generation, Query Processing Engine, etc.
SoC Analyst

Figure 5.1: Using the intelligence translation system with an AI based cyber defense system.

5.1.2 Code Repositories as a Threat Intelligence Source

We mine threat intelligence from issues and bugs raised on web-based hosting service for version control like, GitHub [2], GitLab [3], bitbucket [1], etc. These platforms have been used by developers to host and collaborate on source code development [8]. We extract vulnerabilities raised on these platforms and represent them in a security knowledge graph. The knowledge graph then becomes a store for various vulnerabilities and exposures present in various open source projects, products and libraries (see Figure 5.3). This knowledge can then be queried by various developers helping them create products that are secure from the ground up. Pervasive software security entails cyber supply chain risk management, where the developers are made aware of various threats present in libraries they link with the development of their products.

```
@prefix uco: <http://accl.umbc.edu/ns/ontology/uco#> .

@prefix intel: <http://accl.umbc.edu/ns/ontology/intelligence#> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix xml: <http://www.w3.org/XML/1998/namespace> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix dbp: <http://dbpedia.org/resource#> .

@prefix owl: <http://www.w3.org/2002/07/owl#> .


<Int534962883> a intel:Intelligence ;

intel:hasVulnerability <remote_code_execution> ;


<command_injection> a uco:Means .


<Microsoft_Skype> a uco:Product ;

uco:hasVulnerability <remote_code_execution> ;

owl:sameAs dbp:Skype .


<remote_code_execution> a uco:Vulnerability ;

uco:affectsProduct <Microsoft_Skype> ;

uco:hasMeans <command_injection> ;

owl:sameAs dbp:remote_code_execution .
```

Figure 5.2: RDF for textual input "URL Command Injection Remote Code Execution Vulnerability in Microsoft Skype". Also, $owl : sameAs$ property has been used to augment the data using an external source 'DBpedia' [9].

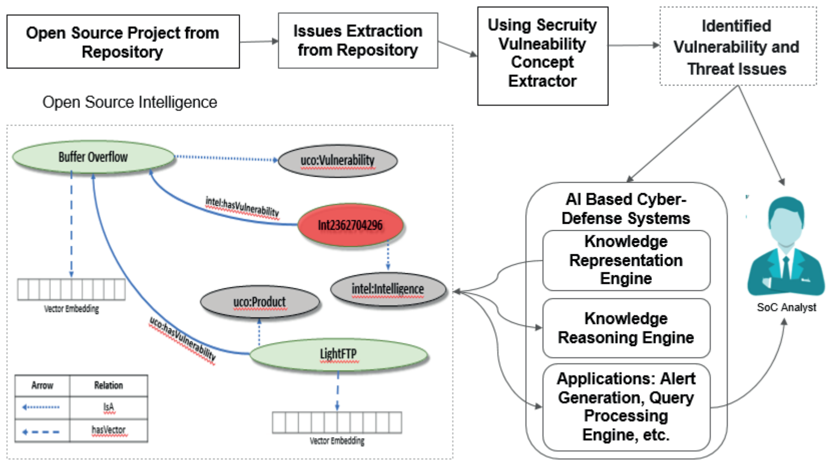
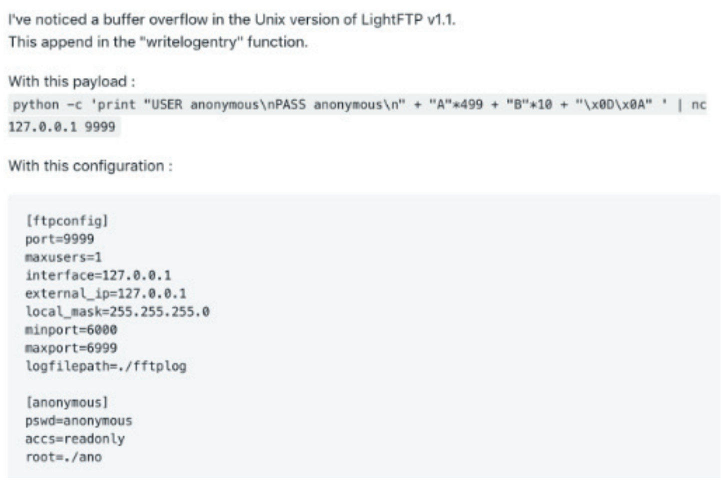

Open Source Project from Repository
Issues Extraction from Repository
Using Secruity Vulneability Concept Extractor
Identified Vulnerability and Threat Issues
Open Source Intelligence
Buffer Overflow
uco:Vulnerability
intel:hasVulnerability
Int2362704296
intel:Intelligence
uco:Product
uco:hasVulnerability
LightFTP
Vector Embedding
Vector Embedding
Arrow
Relation
IsA
hasVector
AI Based Cyber-Defense Systems
Knowledge Representation Engine
Knowledge Reasoning Engine
Applications: Alert Generation, Query Processing Engine, etc.
SoC Analyst

Figure 5.3: Mining threat intelligence from bug and issue reports. Intelligence is stored in a Security Knowledge Graph. The graph represents threat intelligence: *"I've noticed a buffer overflow in the Unix version of LightFTP v1.1"*

Once the SVCE [35] identifies security concepts and entities. We then associated them with Uniform Resource Identifiers (URIs). These URIs are then converted to nodes in our security knowledge graph. We used the Unified Cybersecurity Ontology [71] which integrates heterogeneous data and knowledge schemas from different cybersecurity systems and standards.

We used DBpedia to link various knowledge graph nodes to real world concepts. Entity matching process is performed by using DBpedia [9] and DBpedia spotlight [40]. For example we can use DBpedia to map the string "Adobe Flash" to *dbr:Adobe_Flash*. This external knowledge graph help us map our entities to real world conceptual instances.

After entity linking, we stored the linked data as RDF triples in our security knowledge graph. In our system we need information of cybersecurity intelligence.


I've noticed a buffer overflow in the Unix version of LightFTP v1.1.
This append in the "writelogentry" function.

With this payload :

```
python -c 'print "USER anonymous\nPASS anonymous\n" + "A"*499 + "B"*10 + "\x0D\x0A" ' | nc
127.0.0.1 9999
```

With this configuration :

```
[ftpconfig]
port=9999
maxusers=1
interface=127.0.0.1
external_ip=127.0.0.1
local_mask=255.255.255.0
minport=6000
maxport=6999
logfilepath=./fftplog

[anonymous]
pswd=anonymous
accs=readonly
root=./ano
```

Figure 5.4: Sample issue showing security buffer overflow in a popular Unix FTP client.

Threat intelligence is temporal in nature and may contain other meta-data like, origin, credibility, provenance, etc. UCO though gives us a domain overview of cybersecurity it cannot handle temporal nature of events. So as to handle time in events we use the intelligence ontology [45].

To give an example, Figure 5.5 shows the RDF statements created for the intelligence "*I've noticed a buffer overflow in the Unix version of LightFTP v1.1*" (Figure 5.4). A graphical representation of the same intelligence, 'Int2362704296' has been shown in Figure 5.3. Once we obtain the intelligence in the RDF format, we use it to create the alert and the query system.

5.1.3 Covert Sources

An organization may also have access to various forms of propriety or *covert* data sources. These data sources can also be added as modules, to our Cyber-All-

```
@prefix uco: <http://accl.umbc.edu/ns/ontology/uco#> .
@prefix intel: <http://accl.umbc.edu/ns/ontology/intelligence#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dbp: <http://dbpedia.org/resource#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<Int2362704296> a intel:Intelligence ;
intel:hasVulnerability <buffer_overflow> .

<LightFTP> a uco:Product ;
uco:hasVulnerability <buffer_overflow> ;
owl:sameAs dbp:FTP-server .

<buffer_overflow> a uco:Vulnerability ;
uco:affectsProduct <LightFTP> ;
owl:sameAs dbp:buffer_overflow .
```

Figure 5.5: RDF for textual input "I've noticed a buffer overflow in the Unix version of LightFTP v1.1". Also, $owl : sameAs$ property has been used to augment the data using an external source 'DBpedia' [9].

Intel system. However, in this paper we only describe open data sources.

5.2 System Pipeline & Architecture

Our system pipeline includes a data collection engine that pushes new data into the system from a multitude of data sources. The Cyber-All-Intel system (Figure 5.6) automatically accesses data from some of these sources like NIST's National Vulnerability Database, Twitter, Reddit, Security blogs, dark web markets [15], etc. The system begins by collecting data in a modular fashion from these sources. Followed by a data pre-processing stage where we remove stop words, perform stemming, noun chunking, etc. The data is then stored in a cybersecurity corpus.

After creating a cybersecurity corpus we use a Security Vulnerability Concept Extractor (SVCE) which extracts security related entities and understands their relationships. Our current SVCE [35, 30], trained using various natural language processing techniques, enables us to extract cybersecurity related terms from text, which can then be stored in our knowledge graph. The data is then tagged and vectorized. The tagged entities are then converted to their vector embeddings. These embeddings are then included in the knowledge graph. For more details on the VKG structure see Chapter 4.

The data is asserted in RDF using the Unified Cybersecurity Ontology (UCO) [71]. Ontologies like UCO, Intelligence [45], DBpedia [9], YAGO [68] have been used to provide cybersecurity domain knowledge. The vector part on the other hand was created using a vector generation algorithm (Chapter 5.3.2). An example is shown in Figure 4.2, where we create the VKG structure for the textual input:

Microsoft Internet Explorer allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via a crafted web site, aka "Internet Explorer Memory Corruption Vulnerability."

Triples generated for the above mentioned input text are shown in the Figure 5.9.

The knowledge part of the VKG structure is completed once the triples are added to the system. We used Apache Jena [4] to store our knowledge graph. For example, in Figure 4.2, once added the nodes are linked to the vector embeddings for 'Microsoft_Internet_Explorer', 'remote_attackers', 'execute_arbitrary_code', 'denial_of_service', and 'crafted_web_site'. For our system, we retrain the vector model every two weeks to incorporate the changes in the corpus. We give various details about system execution and evaluation in Chapter 6 and Chapter 7 respectively.

5.3 Cybersecurity Knowledge Improvement

An added benefit of using multiple knowledge representation in the VKG structure is that we can use one representation to improve the other. Improving representation in turn improves the quality of applications that depend on them. In this section we will discuss how we can leverage the vector part of the VKG structure to improve the knowledge graph part and vice versa.

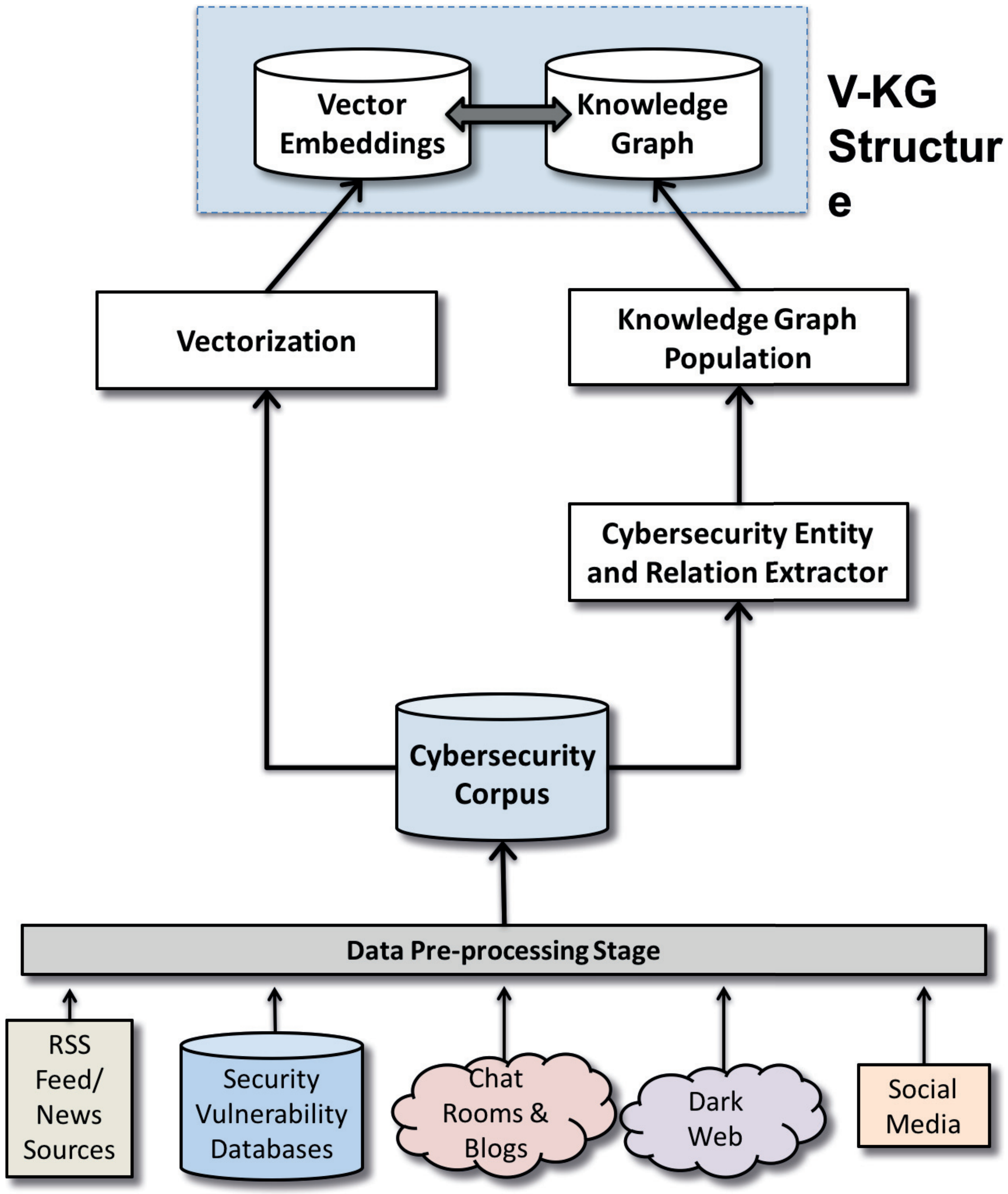


Figure 5.6: Cyber-All-Intel System Architecture.

5.3.1 Improving the Knowledge Graph using Vector Embeddings

The vector representation of different entities can be used by a learning system to enhance the knowledge graph by predicting new relationships between entities. We have created a neural network that takes as input the vector representation of two entities and outputs the relation between them.

This task will help in improving the knowledge graph as in when new data is added. An example task for this agent will be to predict the relation between the entities 'android' and 'buffer overflow', given data from a corpus that have both words. The triple can then be added to the knowledge graph along with their embeddings.

Neural networks can be used to model nonlinear relations between inputs and outputs. We show the different layers of the neural network in Figure 5.7. The figure shows an input layer, multiple hidden layers and an output layer. The neural network has an input layer for embeddings followed by a convolutional layer. After this an activation function (ReLU layer) is added to introduce non-linearity, then a max-pooling layer for down-sampling, followed by a fully connected layer with dropout and softmax output.

Training this network is a supervised learning task. The training set (TS) includes:

$$TS = \{(v_{1,1},\ v_{1,2}, R_1), (v_{2,1},\ v_{2,3}, R_2), ..., (v_{n,k},\ v_{n,l}, R_m)\}$$

where,

$$R_1, R_2, ..., R_m \in R$$

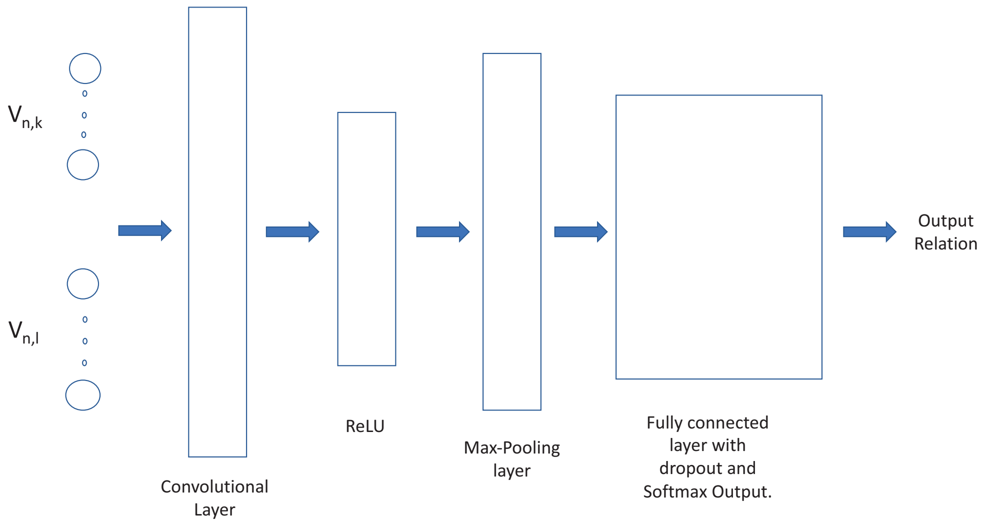


Figure 5.7: Neural network structure to improve knowledge graph using embeddings.

$$k,\, l \in E$$

$$v_{n,k},\, v_{n,l} \in V \,\&\, k \neq l$$

Here R is the set of relations from the UCO ontology [70] from which the output is predicted. $R = [hasProduct,\ hasAttacker,\ hasMeans,\ hasConsequences,\ hasWeakness,\ isUnderAttack,\ hasVulnerability,\ ...]$. V is the set of vector embeddings. E is the set of entity classes. We ensure that input vectors are from different entity classes, i.e. k, l are not same.

Training this network involves minimizing the mean squared error function between the predicted value and the actual relation value.

The triple generated using the two entities and the predicted relation is then added to the knowledge graph part of Cyber-All-Intel. We discuss the performance evaluations for this neural network in Chapter 7.

5.3.2 Improving Vector Embeddings using the Knowledge Graph

In this section we will discuss how we use the knowledge graph part of the VKG structure to improve the vector embeddings. The motivation for this task stems from the need to encode global context present as assertions in the knowledge graph, along with local co-occurring context in vector embeddings. Including the fact that ‘Samsung’ and ‘Apple’ are both mobile phone manufacturers in their vector embeddings will help bring these entities closer in the vector space. This will help improve various applications built upon the structure. The vector embeddings can be used to train machine learning models which will leverage the explicit assertions present in the knowledge graph part.

Current vector generation techniques proposed in [43, 53] provide a method to encode the relational concurrence in a vector space. However, these representations fail to include a global context. Ristoski et al. [59] created *RDF2Vec*, where they adapt neural language models for RDF graph embeddings. They transform the RDF graph data into sequences of entities, which are then considered as sentences. Using these sentences, they train the neural language models to represent each entity in the RDF graph as a vectors.

In our approach, we created a feedforward neural network which takes as input the relational context of an entity, along with the RDF2Vec vector for that entity created using the knowledge graph part of the VKG structure. All the contextual words and the RDF2Vec vector get projected into the same position (as a result of vectors getting averaged). The training criterion is to correctly classify the current

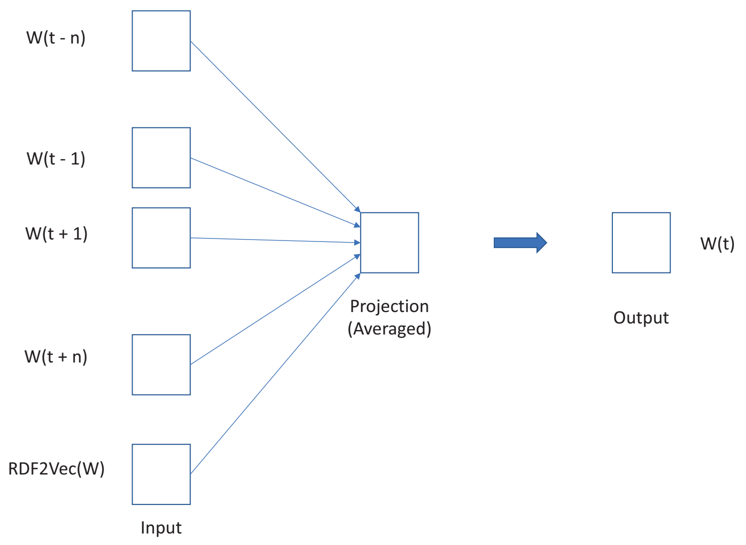


Figure 5.8: Architecture for creating embeddings for $W(t)$ (Here, t is the position of the word). Local context is provided by using co-occurring words $W(t-n)$,...$W(t-1)$, $W(t+1)$,...$W(t+n)$; n is a hyper-parameter used for context window size. Global context is provided by the $RDF2Vec(W)$ vector created using the knowledge graph part.

entity (i.e. the middle word). The output generated serves as the vector encoding for the entity. Figure 5.8 shows the architecture of the neural network. We evaluate these vectors in Chapter 7.

5.4 Applications

In this section we present applications built using the VKG structure.

5.4.1 A Query Processing System

An application running on the VKG structure described in, Section 4 and populated via steps mentioned in Section 5, can handle some specific type of queries. The application can ask a backend query processing engine to list declared entities or relations, search for semantically similar concepts, and compute an output by reasoning over the stored data. This gives us three types of queries, $search$, $list$, and $infer$. These three are some of the basic tasks that an application running on the VKG structure will require, using which we derive our set of query commands (C):

$$C = \{search, list, infer\}$$

A complex query posed by the application can be a union of some of these basic commands. An example query, to the Cyber-All-Intel security informatics system built on our VKG structure can be 'list vulnerabilities in products similar to Google Chrome'; In this query we first have to $search$ for similar products to Google Chrome and then $list$ vulnerabilities found in these products. In a more general setting, an input query can be 'Find similar sites to Taj Mahal, infer their distance from New York'; this includes a $search$ to find similar sites and to $infer$ their distance from New York.

In the VKG structure, knowledge is represented in two parts, in a knowledge graph and in vector space. We argue that a query processing engine developed over the VKG structure should combine the complimentary strengths of knowledge graphs and vector space models. Sub-queries involving similarity search-based tasks will give better and faster results, when carried out on the entire structure. Subqueries that require declarative information retrieval, inference or reasoning should be carried out on the knowledge graph part.

In the query mentioned above, $search$ queries, for the top-k nearest neighborhood search should be performed using the embeddings, and the $list$, $infer$ queries on the knowledge graph part. Domain experts can incorporate various reasoning and inference based techniques in the ontology for the knowledge graph part of the VKG structure. Mittal et al. in their system, CyberTwitter [45] have showcased the use of an inference system to create threat alerts for cybersecurity using Twitter data. Such inference and reasoning tasks can be run on the knowledge graph part of the VKG structure.

Queries most suited for the vector part are those measuring semantic similarity of entities present in the corpus. One of the most simple similarity measures to compute entity similarity is by using the cosine of the angle between entity embeddings. Other query types include scoring, indexing, entity weighting, analogical mapping, nearest / neighborhood search, etc. Turney et al. provide a good survey of various applications and queries that can be built on vector space models [75].

Some form of queries that can't be handled by the vector part are the ones that involve robust reasoning, these include any query that needs to infer logical consequences from a set of asserted triples using inference rules. Such queries need knowledge graph and rule engine support. An argument can be made that in many applications it is enough to come up with a top-K plausible answer set using vector embeddings, however in expert systems like Cyber-All-Intel coming up with more concrete and well reasoned solutions is necessary.

For the Cyber-All-Intel system described in Section 5, some other example queries to the vector part can be, ‘Find products similar to Google Chrome.’, ‘List vulnerabilities similar to buffer overflow’, etc. We evaluate the performance of these queries on different parts in Section 7.

Another advantage provided by integrating vector spaces and knowledge graphs is that we can use both of them to improve the results provided by either of the parts alone. For example (in Figure 4.2), we can use the explicit information provided in the knowledge graph to aid the similarity search in vector space. If we are searching the vector space for entities similar to ‘denial_of_service’, we can further improve our results by ensuring the entities returned belong to class ‘Vulnerability’. This information is available from the knowledge graph. This technique of knowledge graph aided vector space similarity search (*VKG Search*,

Type of queries that are well suited for knowledge graphs include querying the asserted facts for exact values of a triple’s subject, predicate, or object. This information is not present in the vector part explicitly. Knowledge graphs also support an important class of queries that involve semantic reasoning or inference tasks based on rules that can be written in languages like SWRL.

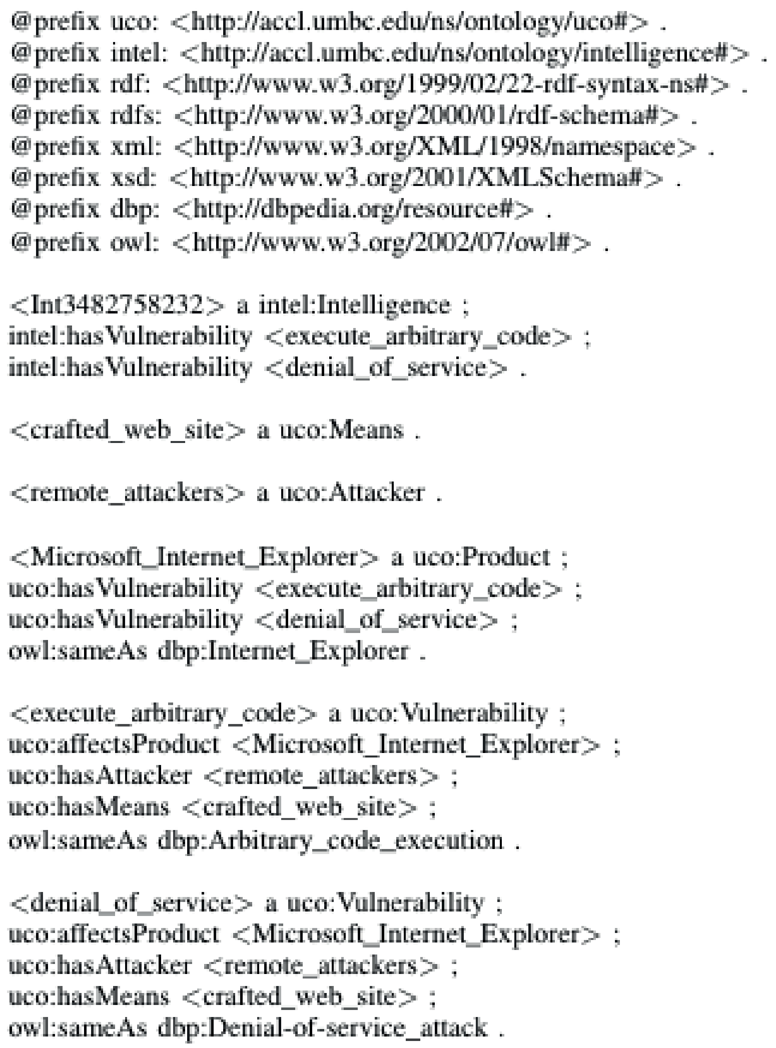
Domain experts can incorporate various reasoning and inference based techniques in the ontology for the knowledge graph part of the VKG structure. Mittal et al. in their system, CyberTwitter [45] have showcased the use of an inference system to create threat alerts for cybersecurity using Twitter data. Such inference and reasoning tasks can be run on the knowledge graph part of the VKG structure.

Knowledge graphs have been used to create various reasoning system where the reasoning logic is provided by the system creator. Hence, we created the $infer$ query which can be used to run application specific reasoning logic as in when required by the input query.

Knowledge graphs also provide the ability to integrate multiple sources of information. We can use the '$owl : sameAs$' relation to integrate other knowledge graphs. Once added these triples and reasoning techniques can be included in our $list$ and $infer$ queries.

In the Cyber-All-Intel system the knowledge graph can handle queries like 'What vulnerabilities are present in Internet Explorer?', 'What products have the vulnerability buffer overflow?', etc. Quality of the output can be improved by running the similarity search on the vector spaces and using the knowledge graph to filter out entities not related to the input entity.

Adding to SPARQL: Our query processing engine aims at extending SPARQL. In SPARQL, users are able to write 'key-value' queries to a database that is a set of 'subject-predicate-object' triples. Possible set of queries to SPARQL are, *Select*, *Construct*, *Ask*, *Describe*, and various forms of *Update* queries. We create a layer above SPARQL to help integrate vector embeddings using our VKG structure. Our query processing engine sends *search* queries to the vector part of the structure, the *list* query to the SPARQL engine for the knowledge graph, and the $infer$ query to the Apache Jena inference engine. Next, we go into the details of our backend query processing system.



```
@prefix uco: <http://accl.umbc.edu/ns/ontology/uco#> .
@prefix intel: <http://accl.umbc.edu/ns/ontology/intelligence#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dbp: <http://dbpedia.org/resource#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<Int3482758232> a intel:Intelligence ;
intel:hasVulnerability <execute_arbitrary_code> ;
intel:hasVulnerability <denial_of_service> .

<crafted_web_site> a uco:Means .

<remote_attackers> a uco:Attacker .

<Microsoft_Internet_Explorer> a uco:Product ;
uco:hasVulnerability <execute_arbitrary_code> ;
uco:hasVulnerability <denial_of_service> ;
owl:sameAs dbp:Internet_Explorer .

<execute_arbitrary_code> a uco:Vulnerability ;
uco:affectsProduct <Microsoft_Internet_Explorer> ;
uco:hasAttacker <remote_attackers> ;
uco:hasMeans <crafted_web_site> ;
owl:sameAs dbp:Arbitrary_code_execution .

<denial_of_service> a uco:Vulnerability ;
uco:affectsProduct <Microsoft_Internet_Explorer> ;
uco:hasAttacker <remote_attackers> ;
uco:hasMeans <crafted_web_site> ;
owl:sameAs dbp:Denial-of-service_attack .
```

Figure 5.9: RDF for textual input "Microsoft Internet Explorer allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via a crafted web site, aka Internet Explorer Memory Corruption Vulnerability". Also, *owl* : *sameAs* property has been used to augment the data using an external source 'DBpedia'.

5.4.1.1 Processing Engine

Let a query proposed by an application to the backend system on the VKG structure be represented by Q^{VKG}. The task of the query processing engine is to run the input query, Q^{VKG}, as efficiently as possible. We evaluate this claim of efficiency in Section 7.1. We do not discuss a query execution plan as multiple expert plans can be generated by domain experts depending on the needs of the application.

As per our need, in the backend system, a query that runs only on the knowledge graph part and only the vector part of the structure are represented as Q^{kg} and Q^{v} respectively. An input query Q^{VKG} can be decomposed to multiple components that can run on different parts namely the knowledge graph and the vector part:

$$Q^{VKG} \rightarrow Q^{kg} \cap Q^{v}$$

An input application query Q^{VKG} can have multiple components that can run on the same part, for example, an input query can have three components, two of which run on the knowledge graph part and the remaining one runs on the vector part. Such a query can be represented as:

$$Q^{VKG} \rightarrow Q^{v} \cap Q_1^{kg} \cap Q_2^{kg} \tag{1}$$

Where, Q_1^{kg} and Q_2^{kg} are the two components that run on the knowledge graph part and Q^{v} component which runs on the vector part.

It is the responsibility of the query processing system to execute these subqueries on different parts and combine their output to compute the answer to the original input query Q^{VKG}. We describe the query execution process using an example.

5.4.1.2 Example query

For the Cyber-All-Intel system an example query issued by the application: ‘Raise an alert if, a vulnerability similar to denial of service is listed in MySQL’, can be considered as three sub-queries which need to be executed on different parts of the VKG structure.

The input query can be considered to be of the type (1), Where the subqueries are:

1. Finding similar vulnerabilities (set - V) to denial of service that will run on the vector embeddings (Q^v).
2. Listing known existing vulnerabilities (set - K) in MySQL (Q_1^{kg}).
3. Inferring if an alert should be raised if a vulnerability (from set V) is found in the product MySQL. This sub-query will run on the knowledge graph part (Q_2^{kg}).

The query can be represented as:

$$\begin{aligned} Q^{VKG} = \{\{search, `denial\_of\_service', V\} \cap \\ \{list, vulnerability, `MySQL', K\} \cap \\ \{infer, V, K, `MySQL', alert\}\} \quad \text{(Query 1)} \end{aligned}$$

The query execution plan for (Query 1) is to first run Q^v and Q_1^{kg} simultaneously and compute the sets V and K. After computing the sets the engine is supposed to run Q_2^{kg}.

The first part of the input query (Query 1), is of the form Q^v and will run on the vector part of the VKG structure. Its representation is:

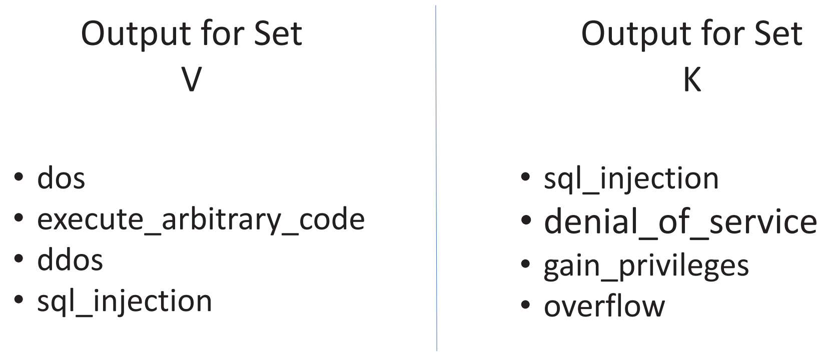
$$Q^v = \{search, `denial\_of\_service', V\}$$

The output generated is a set V (Figure 5.10) and contains vulnerabilities similar to 'denial_of_service'. We used the VKG search to compute this set and filter out all non vulnerabilities. The set V will be utilized by other subqueries (Q_2^{kg}) to generate it's output.

The second part of the input query (Query 1) is the first sub-query to run on the knowledge graph part of the VKG structure.

$$Q_1^{kg} = \{list, vulnerability, `MySQL', K\}$$

The goal of this query is to list all vulnerabilities (Figure 5.10) present in 'MySQL' that are explicitly mentioned in the knowledge graph (set - K).



Output for Set V

- dos
- execute_arbitrary_code
- ddos
- sql_injection

Output for Set K

- sql_injection
- denial_of_service
- gain_privileges
- overflow

Figure 5.10: The output of the sub-queries Q^v and Q_1^{kg} when run on the Cyber-All-Intel System. As there is some overlap between the sets V and K the output for the subquery Q_2^{kg} will be 'Alert = Yes'

The third part of the input query (Query 1) is the second subquery to run on the knowledge graph part of the VKG structure.

$$Q_2^{kg} = \{infer, V, K, `MySQL', alert\}$$

Here, the output is to reason whether to raise an 'alert' if some overlap is found between the sets V & K. Query Q_2^{kg} requires an inference engine to output an alert based on some logic provided by domain experts or system security administrators. In Figure 5.10 as there is overlap between the sets V and K an alert will be raised.

5.4.2 Knowledge Augmentation and Alerts

In the field of cybersecurity a security analysts need to be aware of all possible threats and vulnerabilities to their cyber-infrastructure. We have created an intelligence alert system on top of the VKG structure, which briefs an analyst about various threats relevant to the software and hardware components present in an enterprise, when intelligence from multiple sources is analyzed and aggregated.

In the past we created, *CyberTwitter* [45] to issue alerts about vulnerabilities found in various products used by an organization. The CyberTwitter system uses a knowledge graph where reasoning was done by adding SWRL rules. In the Cyber-All-Intel system, the SWRL rules have been extended and we also investigate similar products using the vector space to issue alerts based on an organization's 'system profile'.

In this section we discuss two things, firstly, how we augment the knowledge graph with other sources of information and secondly, how we generate alerts.

5.4.2.1 Knowledge Augmentation

Many a times a query can come in that requires more knowledge for the answer to be computed than what is present in the text corpus used for training the VKG structure. An example query like "What products similar to Internet Explorer are produced by Google Inc.?" Such a query needs to first compute the set of possible products similar to 'Internet Explorer' and then filter out the ones that are not produced by the entity 'Google Inc.'.

Knowledge graph nodes in the VKG structure can be used to add information from other sources like DBpedia [9], YAGO [68], etc. This helps integrate information that is not present in the input corpus. Along with these sources we can add more information gathered from local organizational structure like network activity, shared library dependencies of a program executable, etc. This knowledge helps in adding local organizational knowledge to the system.

For our system to handle these type of queries we used the information present in existing knowledge graphs that were populated using other textual sources and techniques. As a proof of concept we integrated our *Cyber-All-Intel* VKG structure's knowledge graph part with DBpedia [9]. During this process of integration we asserted various products and vulnerabilities with their counterparts in the DBpedia knowledge graph. Figure 5.9 shows an example where the property *owl : sameAs* is used to assert counterparts for VKG instances of 'Microsoft_Internet_Explorer', 'Arbitrary_code_execution', and 'Denial-of- service_attack'.

To include some local knowledge from the system that needs to be protected, we add shared library dependencies of programs installed. This information was collected using an Ubuntu system using the 'objdump' tool[2] and filtering out library dependencies using the 'NEEDED' flag. The dependencies for installed software were then asserted in a knowledge graph.

Adding both global and local information to the knowledge graph helps us in improving the quality of alerts and recommendations.

5.4.3 Generating Alerts

For our alert system we create vector embeddings (for the VKG structure) using the augmented knowledge graph (with DBpedia and library dependencies of programs installed) discussed in Section 5.4.2.1, and the generation method mentioned in Section 5.3.2. We extend the SWRL rules included in the CyberTwitter [45] system.

We also ask the security analyst to provide the recommender system a 'system profile'. The profile contains information about the operating system, various installed softwares and their version information. We use the profile information as part of our rules to generate alerts.

We created a VKG based alert system that has two logical parts:

1. *Vulnerability Alerts using factual data:* We created a rule based system to issue alerts using the knowledge graph part of the VKG structure which includes factual data like, collected intelligence, DBpedia, library dependencies

[2]`https://sourceware.org/binutils/docs/binutils/objdump.html`

of installed programs, etc.

We utilized SWRL rules to include a reasoning engine analogous to a deductive based approach that a security analyst might take to figure out threats to her system. SWRL rules contain two parts, antecedent part (body), and a consequent (head). Informally, a rule may be read as meaning that if the antecedent holds (is "true"), then the consequent must also hold.

We have modified and generalized CyberTwitter's SWRL based recommendation engine [45] (See Chapter 3), where we first compute if an intelligence is 'valid and current' and in the second part we use a valid intelligence to raise an alert if its in the analyst's system profile.

2. *Vulnerability Alerts for similar products:* It is also necessary for the system to look for similar products that might also be at risk (The analyst can choose whether she wants these alerts). To keep a security analyst updated we also have to consider possible vulnerabilities that may exist in products that share library dependencies and/or developed by similar companies. For example, in case of products like Mozilla Firefox and Thunderbird, which are developed by the same company and have a considerable overlap in library dependencies; an alert generated for one, warrants an investigation into the other.

   So as to create such alerts we leverage the vector part of the VKG structure. When we get an alert about a vulnerability in a product from the factual data using the SWRL rules mentioned above, we look into possible intelligence obtained for products in the neighborhood of the vulnerable product and re-

reason the SWRL rules with added information. We factor in the number of shared library dependencies, developing companies, etc in the SWRL rules. This vector neighborhood was created using the augmented knowledge graph mentioned in Section 5.4.2.1.

After looking at the alerts generated using the factual data and by investigating similar products, we then push these alerts to the analyst depending on the organization's 'system profile'. We evaluate our recommender and alert system in Section 7.

Once we populated our security knowledge graph with the information about installed software and linked dependencies (see Section 5.4.4) and also add, threat intelligence mined from issues and bug reports, we utilized it to create an alert generation system and a query system.

5.4.4 Programming Environment Augmentation System

We create a system that will inform a developer about potential threats and vulnerabilities that the product in development might inherit as a result of linking to a vulnerable open source project or library. We mine threat intelligence from issues and bugs raised on web-based hosting service for version control like, GitHub [2], GitLab [3], bitbucket [1], etc. These platforms have been used by developers to host and collaborate on source code development [8]. We extract vulnerabilities raised on these platforms and represent them in a security knowledge graph. The knowledge graph then becomes a store for various vulnerabilities and exposures present in various open source projects, products and libraries (see Figure 5.3). This knowledge can then be queried by various developers helping them create products that are secure from the ground up. Pervasive software security entails cyber supply chain risk management, where the developers are made aware of various threats present in libraries they link with the development of their products.

We also create another application that can track installed software on a client machine, and then use the above mentioned knowledge graph to *reason* alerts for the security analyst. These alerts warn the analyst if an installed software is linked to a vulnerable open source library or project. We built a proof of concept for this application that runs on a linux installation.

The developer before linking to an open source library or using a project should be able to query the security knowledge graph to check for known vulnerabilities. We have created a SPARQL[3] endpoint that can accept queries which run on our knowledge graph. An example query to list all vulnerabilities in *LightFTP*:

```
SELECT ?y WHERE {

?LightFTP <hasVulnerability> ?y .

}
```

An example query to look up vulnerabilities in linked libraries to the installed application *firefox*:

[3] https://www.w3.org/TR/rdf-sparql-query/

```
SELECT ?x WHERE {

?firefox <Is_Linked_to> ?z

?z <hasVulnerability> ?x.
}
```

5.4.4.1 Alert Generation System

The system will reason on our security knowledge graph and generate alerts. In our system we include SWRL rules[4] so as to generate alerts. SWRL rules contain two parts, antecedent part (body), and a consequent (head). The body and head consist of conjunctions of a set of 'atoms'. Informally, a rule may be read as meaning that if the antecedent holds (is "true"), then the consequent must also hold. For our system we see two potential alert scenarios:

1. A developer is linking to a library or a project with known vulnerabilities and threats: The system will take in all the libraries that a developer wants to use and then trigger an alert if it finds a vulnerability or threat in any one of these libraries. We see this as a developer initiated scenario.

   Our alert system also checks other linked libraries that link to the ones mentioned by the developer. Some example rules included in our system:

   ```
           Rule for vulnerable project utilization:
   ```

[4]https://www.w3.org/Submission/SWRL/

```
Product(?x)^ Utilizes(?x, ?y)^
hasVulnerability(?y, ?z)


==>


RaiseAlert(?x,”Yes”)


Rule for linked library vulnerability check:


Product(?x)^
Is_Linked_To(?x, ?y)^
hasVulnerability(?y, ?z)


==>


RaiseAlert(?x,”Yes”)


Rule for secondary linked library vulnerability
check:


Product(?x)^
```

```
Is_Linked_To(?x, ?y)^
s_Linked_To(?y, ?z)^
hasVulnerability(?z, ?u)


==>


RaiseAlert(?x,"Yes")
```

The rules raise an alert if any vulnerability or threat is found in linked libraries and projects. For the first rule above 'Rule for vulnerable project utilization', given a product node $?x$, the rules check for edge: $Utilizes(?x, ?y)$, to hop to the graph node $?y$. Once at $?y$ it checks for the edge: $hasVulnerability(?y, ?z)$ to hop to node $?z$. If the above node exists an alert is generated. A similar technique is used to evaluate other rules mentioned above.

2. An installed application on a client machine is linked to compromised dependencies: This is an information triggered alert, where influx of new threat intelligence warrants a lookup for vulnerable installed software. The system should automatically inform a security analyst that an installed application on a client machine is vulnerable. An example rule for this alert:

```
Rule for vulnerable libraries:


Library(?x)^
```

```
hasVulnerability(?x, ?y)^
Is_Linked_To(?z, ?x)


==>


RaiseAlert(?z,"Yes")


Rule for vulnerable projects:


Project(?x)^
hasVulnerability(?x, ?y)^
Utilizes(?z, ?x)


==>


RaiseAlert(?z,"Yes")
```

Chapter 6

Experimental Setup

6.1 Cyber-All-Intel System

For Cyber-All-Intel system, we created a Cybersecurity corpus as discussed in Chapter 5 and shown in Figure 5.6. Data for the corpus is collected from many sources, including chat rooms, dark web, blogs, RSS feeds, social media, and vulnerability databases. The current corpus has 85,190 common vulnerabilities and exposures from the NVD dataset maintained by the MITRE corporation, 351,004 cleaned Tweets collected through the Twitter API, 25,146 Reddit and blog posts from sub-reddits like, r/cybersecurity/, r/netsec/, etc. and a few dark web posts [15].

For the vector space models, we created embeddings by setting vector dimensions as: 500, 1000, 1500, 1800, 2500 and term frequency as: 1, 2, 5, 8, 10 for each of the dimensions. The context window was set at 7. The knowledge graph part was created using the the steps mentioned in Chapter 5 and the VKG structure was generated by linking the knowledge graph nodes with their equivalents in the vector model vocabulary (see Chapter 4).

In order to conduct various evaluations, we first created an annotated test set. We selected some data from the cybersecurity corpus and had it annotated by a group of five graduate students familiar with cybersecurity concepts. The annotators were asked to go through the corpus and mark the following entity classes: *Address, Attack / Incident, Attacker, Campaign, Attacker, CVE, Exploit, ExploitTarget, File, Hardware, Malware, Means, Consequence, NetworkState, Observable, Process, Product, Software, Source, System, Vulnerability, Weakness*, and *VersionNumber*. They were also asked to annotate various relations including *hasAffectedSoftware, hasAttacker, hasMeans, hasWeakness, isUnderAttack, hasSoftware, has CVE_ID*, and *hasVulnerability*. These classes and relations correspond to various classes and properties listed in the Unified Cybersecurity Ontology and the Intelligence Ontology [45]. For the annotation experiment, we computed the inter-annotator agreement score using the Cohen's Kappa [65]. Only the annotations above the agreement score of 0.7 were kept.

The annotators were also tasked to create sets of similar products and vulnerabilities so as to test various aspects of the Cyber-All-Intel system. The most difficult task while designing various experiments and annotation tasks was to define the meaning of the word 'similar'. Should similar products have the same vulnerabilities, or same use? In case of our cybersecurity corpus we found that the two sets, same vulnerabilities and same use were co-related. For example, if two products have SQL injection vulnerability we can say with certain confidence that they use some form of a database technology and may have similar features and use. If they have Cross-Site Request Forgery (CSRF) vulnerability they may generally belong to the product class of browsers.

Annotators manually created certain groups of products like, operating systems, browsers, databases, etc. OWASP[1] maintains groups of similar vulnerabilities[2] and attacks[3]. We created 14 groups of similar vulnerabilities, 11 groups of similar attacks, 31 groups of similar products. A point to note here is that, in many cases certain entities are sometimes popularly referred by their abbreviations, we manually included abbreviations in these 56 groups. For example, we included DOS and CSRF which are popular abbreviations for Denial Of Service and Cross-Site Request Forgery respectively in various groups.

6.2 Programming Environment Augmentation System

In order to evaluate the system and collect empirical data we ran our system under experimental conditions. The system was run on a Ubuntu[4] Linux installation with 81 installed programs, some of these were pre-installed. We extracted the library and project dependencies for these 81 installed programs and represent this information in our security knowledge graph (see Chapter 5).

We collected 110,800 issues posted on GitHub [2], using the GitHub Rest API. For our experiments and to create a valid proof of concept, we limit issue collection to the GitHub repositories for the 81 installed projects. We also use only the issues posted after January 2018 in our analysis. Out of the 110,800 issues collected our SVCE (see Chapter **??**) filtered 9,194 security issues. We then assert these security vulnerabilities in our knowledge graph (see Chapter 5). Figure 5.5, lists triples

[1]`https://www.owasp.org/index.php/Main_Page`

[2]`https://www.owasp.org/index.php/Category:Vulnerability`

[3]`https://www.owasp.org/index.php/Category:Attack`

[4]`https://www.ubuntu.com/`

generated for a popular FTP client.

We performed an initial evaluation of our prototype system using the bug-reports collected. We evaluate the quality of the tags generated by the SVCE module, and how often our system missed intelligence because it discarded relevant details. We did not evaluate our entity matching process as it was done through DBpedia APIs. Human assessments and annotation was done by students familiar with the cybersecurity domain.

For our first evaluation measure we check the quality of tags generated by our SVCE module. We tagged 150 randomly selected security issues and then manually checked the tags. The annotators had to evaluate if the SVCE output was correct, partially correct or wrong. Our annotators agreed on the fact that 98 issues were marked correctly by the SVCE module and out of the remaining 52 issues, 18 were tagged completely wrong and the remaining were tagged partially correct. The annotators were then asked to look into the alerts generated for the 98 issues correctly identified, the system raised appropriate alerts for each.

We evaluated the loss of intelligence because of discarded issues, i.e., those not included in the dataset of 9,194 security issues. A random sample of 200 issues was generated from the discarded issues. In these, our annotators found 9 issues with actionable security related information. We believe that these were wrongfully tagged by our SVCE module because of spelling mistakes, unidentifiable characters, informal slang expressions, non-English words, etc.

Chapter 7

Results & Discussion

7.1 Evaluations

Cyber-All-Intel is a threat intelligence system which aims to provide tactical and operational support to the security analyst. The goal of the system is to add value to the analyst's work flow and enable her to make efficient security policy decisions. The system aims to reduce the 'cognitive load' on the security analyst. To ensure that Cyber-All-Intel is able to sufficiently aid the analyst, we first evaluate the system by questioning it's core utilities. What is the quality of the information that is being provided by the system? Such an information can be about an attack or a vulnerability.

The second method to evaluate our system is to measure how it can help a security analyst keep an updated policy for her organization. The system can provide the analyst with various similarities and differences between various variants of attacks. For example, the system can provide the analyst with the differences between 'WannaCry' and 'notPetya'. This information can then allow an analyst to create specific policy updates that help protect the organization.

We also evaluate the knowledge improvement, query processing engine, alert generation capabilities of Cyber-All-Intel.

7.1.1 Evaluating core capabilities

So as to evaluate the core capabilities of the system we focus on two features, first, the quality of new intelligence obtained or updates made to existing intelligences. Second, to evaluate if the system is able to highlight the similarities and differences between various attacks and vulnerabilities.

For the first one, we leverage the annotators mentioned in Chapter 6. We provided them with the VKG structure generated along with the text that was used to generate it. For example, we provide an annotator with the VKG structure of 'WannaCry' along with the text from our cybersecurity corpus that relates to WannaCry. The annotators were then asked to check if the VKG structure created was correct. We gave the annotators 60 such attacks, 49 of these were marked correct. Each attack was annotated by at least 2 annotators.

In the second one, we provided the annotators with pairs of attacks and vulnerabilities which are similar, as measured by comparing their VKG embeddings. We also gave them a policy to prevent one of the attacks. Their task was to modify the given policy so that it is able to protect an organization from both. For example, we provided our annotators with the VKG structure for 'WannaCry' and 'notPetya', along with a policy to prevent WannaCry. The annotators were then tasked to change the policy so that it can prevent both WannaCry and notPetya. We ran this experiment for 22 such pairs, along with the policies to prevent one of the attacks present in the pair. Each pair was annotated by at least 2 annotators. Of the given 22 pairs, the annotators were able to correctly modify 18 policies.

Such capabilities add value to the analyst workflow. Providing this information can help the analyst make informed policy level changes. We would also like to bring to notice a possible feature, where an AI system will automatically suggest policy level changes to the security analyst. Research on this feature is ongoing but such a capability will be built using the core capabilities of the Cyber-All-Intel system.

What is missing in existing proprietary SIEMs like LogRhythm, Splunk, IBM QRadar, and AlienVault, etc. is the integration of threat intelligence from disparate sources followed by efficient interpretation and reasoning on data using known intelligence [78, 79]. This can reduce false positives and improve the current state of the art in this domain. Also, it reduces the cognitive load on the analyst, because the system can fuse threat intelligence with observed data to detect attacks early, ideally left of exploit.

7.1.2 Evaluating knowledge improvement

In Cyber-All-Intel we leverage different knowledge representations in our VKG structure. The knowledge graph part is designed to hold more global context. The vector space embeddings on the other hand have been created using the local context around the entity. In Chapter 5.3, we discuss our methodology to leverage the different parts of the VKG to improve each other.

In the knowledge graph improvement task discussed in Chapter 5.3.1 we classify the relation between two entities using a deep learning model. The model was trained on the true positive relations explicitly declared in 60,000 CVEs[1]. 30,000

[1]There were about 95,000 CVEs when this article was published.

were used as test set. Our model has an accuracy of 81.5%.

To evaluate the quality vector embeddings generated in Chapter 5.3.2 we use the sets of 'similar' products and vulnerabilities discussed in Chapter 6. The task was to evaluate if similar real-word products and vulnerabilities are present in the same neighborhood in the vector space. For the task, we query the embeddings on one of the elements in the similar annotated sets and then compare the entities of the set returned (We compare products only to products, vulnerability only to vulnerability,...). For the vector space with dimensionality of 1500 and term frequency 2, we get a precision of 0.84 and recall of 0.24.

7.1.3 Evaluating the query processing engine

Using the data and annotation test sets mentioned in Chapter 6, we evaluate our query processing engine. In Chapter 5.4.1.1, we describe our query processing engine and its three query commands: *search*, *list*, and *infer*. Here we evaluate *search* and *list* query but not the *infer* queries as they depend on the reasoning logic provided in the ontology and can vary with application.

7.1.3.1 Evaluating the *search* query

An input query to the VKG structure to find similar concepts can either run on the vector space using various neighborhood search algorithms [23, 34] or the knowledge graph using ontology matching, ontology alignment, schema matching, instance matching, similarity search, etc. [63, 20, 84]. To evaluate the vector embeddings part of the VKG structure we used the 'similar' sets created by the annotators. We trained various vector space models with vector dimension, 500, 1000, 1500, 1800, 2500 and term frequency, 1, 2, 5, 8, 10. Increasing the value of dimensionality and decreasing the term frequency almost exponentially increases the time to generate the vector space models. We first find the combination of parameters for which the Mean Average Precision (MAP) is highest, so as to use it in comparing the performance of vector space models with knowledge graphs and graph aided vectors in the VKG structure, in finding similar vulnerabilities, attacks, and products. For the 56 similar groups the vector model with dimensionality of 1500 and term frequency 2, had the highest MAP of 0.69 (Figure 7.1). Models with higher dimensions and word frequency performed better.

To compare the performance of the *search* query over vector space and its counterpart from the knowledge graph side we used the vector embedding model with dimensionality of 1500 and term frequency 2. To compute instance matching on knowledge graphs, we used an implementation of ASMOV (Automated Semantic Matching of Ontologies with Verification) [29].

On computing the MAP for both vector embeddings and the knowledge graphs we found that embeddings constantly outperformed the knowledge graphs. Figure 7.2, shows that the MAP value for vector embeddings was higher 47 times out of 56 similarity groups considered. The knowledge graph performed significantly bad for vulnerabilities and attacks as the structural schema for both attacks and vulnerabilities was quite dense with high number of edges to different entities. This significantly affected the performance of schema matching.

To test the advantages of our VKG structure we evaluate the VKG search (see Chapter 5.4.1.1) against the vector space model. The VKG search on vector space achieved a MAP of 0.8, which was significantly better than the MAP score (0.69) achieved by using just the vector model. The reason for higher quality results obtained by using the VKG search is due to the fact that we can filter out entities by using class type declarations present in the knowledge graph.

Model	Graphs	Vectors	VKG Search
MAP	0.43	0.69	0.80

Table 7.1: Best Mean Average Precision for knowledge graphs, vector space models, and VKG structure.

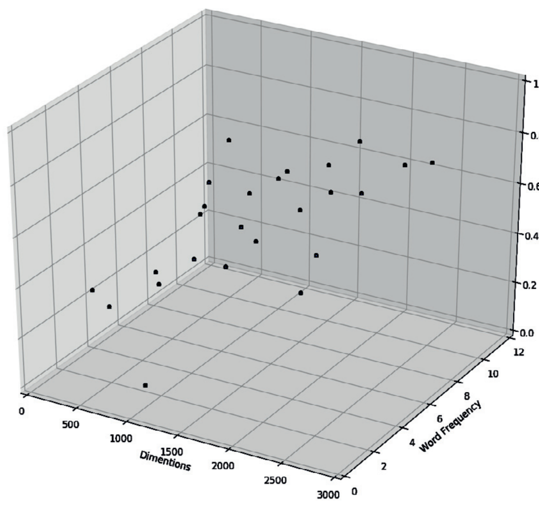


Figure 7.1: Mean Average Precision for different dimensions and word frequency. Models with higher dimensions and word frequency performed better.

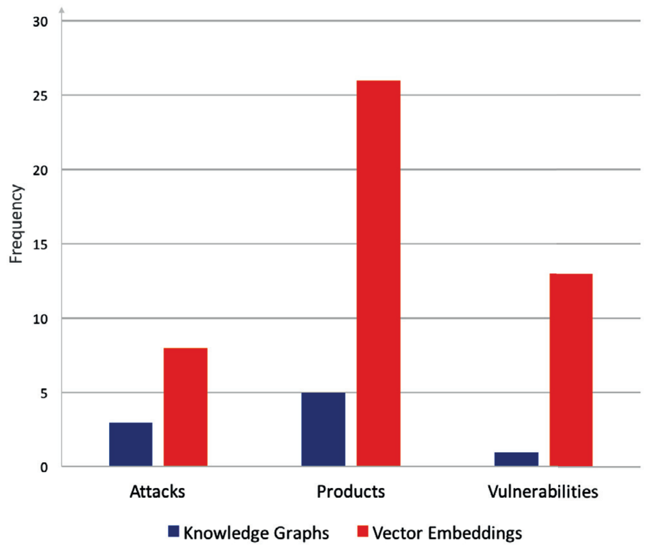


Figure 7.2: The number of times the MAP score was higher for the two knowledge representation techniques for the 56 similar groups. Vector embeddings performed better than knowledge graphs. Embeddings performed better in 8 attacks, 26 products, and 13 vulnerabilities.

7.1.3.2 Evaluating the *list* query

Since the declarative assertions are made in the VKG's knowledge graph, to evaluate the *list* query we evaluate the quality of the knowledge graph part. The *list* query can not be executed on the vector space as there is no declarative information in embeddings.

To check the quality of the knowledge graph triples generated from the raw text we asked the same set of annotators to manually evaluate the triples created and compare them with the original text. The annotators were given three options, correct, partially correct, and wrong. From 250 randomly selected text samples from the cybersecurity data, the annotators agreed that 83% were marked correct, 9% were partially correct, and 8% were marked wrong.

7.1.4 Evaluating the alert system

In Chapter 5.4.2 we discussed our alert system. In our system we first generate alerts using the factual data obtained by augmenting incoming intelligence with shared library dependencies and DBpedia linkages. This data is then used by a rule based system to generate alerts. Once we get an alert about a product, we also investigate other products in it's vector neighborhood created using the augmented knowledge graph. Alerts are then pushed to the analyst depending on the organization's 'system profile'.

To evaluate the quality of these alerts we conducted a small user study where we asked five assessors to judge the usefulness of alerts (options: useful, maybe, useless) given the set of sources responsible for the alert. Out of 55 alerts generated 43 were marked as useful, 3 were marked useless, and the remaining 9 were marked as maybe.

7.1.5 Evaluating the Programming Environment Augmentation System

In order to evaluate the system and collect empirical data we ran our system under experimental conditions. The system was run on a Ubuntu[2] Linux installation

[2]`https://www.ubuntu.com/`

with 81 installed programs, some of these were pre-installed. We extracted the library and project dependencies for these 81 installed programs and represent this information in our security knowledge graph (see Chapter 5). Figure **??**, shows the triples generated for a popular installed software.

We collected 110,800 issues posted on GitHub [2], using the GitHub Rest API. For our experiments and to create a valid proof of concept, we limit issue collection to the GitHub repositories for the 81 installed projects. We also use only the issues posted after January 2018 in our analysis. Out of the 110,800 issues collected our SVCE filtered 9,194 security issues. We then assert these security vulnerabilities in our knowledge graph (see Chapter 5). Figure 5.5, lists triples generated for a popular FTP client.

We performed an initial evaluation of our prototype system using the bug-reports collected. We evaluate the quality of the tags generated by the SVCE module, and how often our system missed intelligence because it discarded relevant details. We did not evaluate our entity matching process as it was done through DBpedia APIs. Human assessments and annotation was done by students familiar with the cybersecurity domain.

For our first evaluation measure we check the quality of tags generated by our SVCE module. We tagged 150 randomly selected security issues and then manually checked the tags. The annotators had to evaluate if the SVCE output was correct, partially correct or wrong. Our annotators agreed on the fact that 98 issues were marked correctly by the SVCE module and out of the remaining 52 issues, 18 were tagged completely wrong and the remaining were tagged partially correct. The annotators were then asked to look into the alerts generated for the 98 issues correctly identified, the system raised appropriate alerts for each.

We evaluated the loss of intelligence because of discarded issues, i.e., those not included in the dataset of 9,194 security issues. A random sample of 200 issues was generated from the discarded issues. In these, our annotators found 9 issues with actionable security related information. We believe that these were wrongfully tagged by our SVCE module because of spelling mistakes, unidentifiable characters, informal slang expressions, non-English words, etc.

Chapter 8

Conclusion & Future Work

This thesis presents *CyberTwitter* & *Cyber-All-Intel* systems. Both these systems try to represent cyber threat intelligence in various representation techniques. Cyber-All-Intel is a system for knowledge extraction, representation and analytics in an end-to-end pipeline grounded in the cybersecurity informatics domain. The system creates a cybersecurity corpus by collecting threat and vulnerability intelligence from various textual sources like, national vulnerability databases, dark web vulnerability markets, social networks, blogs, etc. which are then represented as instances of our VKG structure.

The Cyber-All-Intel system also pro-actively tries to improve the underlying cybersecurity knowledge. We have created neural network models, that take the vector part of the VKG structure and improves the knowledge graph. The knowledge graph part serves as the input to the vector generating part, adding more global knowledge to these embeddings.

We use the system to answer complex cybersecurity informatics queries and issue alerts to the system analyst. Some other applications that can be added in the future are: suggestions for policy updates, linking an organization's in-network and endpoint sensors to create a robust Intrusion Detection and Prevention System (IDPS), etc.

These extensions and planned future work, brings us closer to our main aim - creating an artificial intelligence system to aid the security analyst.

Bibliography

[1] Bitbucket. https://bitbucket.org.

[2] Github. https://github.com.

[3] Gitlab. https://about.gitlab.com/.

[4] Jena apache fuseki: serving rdf data over http,. `http://jena.apache.org/documentation/serving_data/index.html`.

[5] Owl web ontology language. `http://www.w3.org/TR/owl-features/`.

[6] Resource description framework (rdf). `http://www.w3.org/RDF/`.

[7] Sparql protocol and rdf query language 1.1 overview. `http://www.w3.org/TR/sparql11-overview/`.

[8] Github predicts hottest 2018 open source trends, Feb 2018.

[9] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. *DBpedia: A Nucleus for a Web of Open Data*. Springer, 2007.

[10] Sean Barnum. Standardizing cyber threat intelligence information with the structured threat information expression (stix^{TM}). *MITRE Corporation, July*, 2012.

[11] BBC. 'NSA malware' released by Shadow Brokers hacker group. `http://www.bbc.com/news/technology-39553241`, 2017. [Online; accessed 2-March-2018].

[12] Tim Berners-Lee, Tim Bray, Dan Connolly, Paul Cotton, Roy Fielding, Mario Jeckle, Chris Lilley, Noah Mendelsohn, David Orchard, Norman Walsh, and Stuart Williams. Uniform resource identifier, December 2004.

[13] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *The Scientific American*, 2001.

[14] Johan Bollen, Huina Mao, and Xiao-Jun Zeng. Twitter mood predicts the stock market. *CoRR*, abs/1010.3003, 2011.

[15] Gwern Branwen, Nicolas Christin, David Décary-Hétu, Rasmus Munksgaard Andersen, StExo, El Presidente, Anonymous, Daryl Lau, Sohhlz, Delyan Kratunov, Vince Cakic, Van Buskirk, and Whom. Dark net market archives, 2011-2015. `https://www.gwern.net/DNM%20archives`, July 2015.

[16] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[17] Danica Damljanovic, Johann Petrak, Mihai Lupu, Hamish Cunningham, Mats Carlsson, Gunnar Engstrom, and Bo Andersson. Random indexing for finding similar nodes within large rdf graphs. In *Extended Semantic Web Conference*, pages 156–171. Springer, 2011.

[18] Randall Davis, Howard Shrobe, and Peter Szolovits. What is a knowledge representation? *AI magazine*, 14(1):17, 1993.

[19] Bertrand De Longueville, Robin S Smith, and Gianluca Luraschi. Omg, from here, i can see the flames!: a use case of mining location based social networks to acquire spatio-temporal data on forest fires. In *Proceedings of the 2009 international workshop on location based social networks*, pages 73–80. ACM, 2009.

[20] Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. A similarity measure for approximate querying over rdf data. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 205–213, New York, NY, USA, 2013. ACM.

[21] dhs. Stop.think.connect. multilingual resources. `https://www.dhs.gov/stopthinkconnect-multilingual-resources/`, 2015.

[22] Timothy W. Finin, Dawn J. Lawrie, James Mayfield, and Paul McNamee. Hltcoe participation in tac kbp 2016: Cold start and edl. 2016.

[23] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[24] Aditi Gupta, Hemank Lamba, Ponnurangam Kumaraguru, and Anupam Joshi. Faking sandy: characterizing and identifying fake images on twitter during hurricane sandy. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 729–736. International World Wide Web Conferences Steering Committee, 2013.

[25] Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. *arXiv preprint arXiv:1506.01094*, 2015.

[26] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[27] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml, May 2004.

[28] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Com-*

putational Linguistics: Long Papers-Volume 1, pages 873–882. Association for Computational Linguistics, 2012.

[29] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology matching with semantic verification. *Web semantics*, 7 3:235–251, 2009.

[30] Arnav Joshi. A linked data resource for software security concepts and vulnerability descriptions. Master's thesis, University of Maryland, Baltimore County, August 2013.

[31] Arnav Joshi, Ravendar Lal, Tim Finin, and Anupam Joshi. Extracting cybersecurity related linked data from text. In *Proceedings of the 7th IEEE International Conference on Semantic Computing*. IEEE Computer Society Press, September 2013.

[32] Michael Kandefer, S Shapiro, Adam Stotz, and Moises Sudit. Symbolic reasoning in the cyber security domain. 2007.

[33] Judith L. Klavans. Cybersecurity - what's language got to do with it? 2015.

[34] Saar Kuzi, Anna Shtok, and Oren Kurland. Query expansion using word embeddings. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1929–1932, New York, NY, USA, 2016. ACM.

[35] Ravendar Lal. Information Extraction of Security related entities and concepts from unstructured text. Master's thesis, University of Maryland, Baltimore County, May 2013.

[36] G. Lawton. Open source security: opportunity or oxymoron? *Computer*, 35(3):18–21, Mar 2002.

[37] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, pages 2181–2187, 2015.

[38] Przemyslaw Maciolek and Grzegorz Dobrowolski. Cluo: Web-scale text mining system for open source intelligence purposes. *Computer Science (AGH)*, 14:45–62, 2013.

[39] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[40] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. DBpedia spotlight: shedding light on the web of documents. In *7th Int. Conf. on Semantic Systems*, pages 1–8. ACM, 2011.

[41] Microsoft. Microsoft Security Bulletin MS17-010 - Critical-Security Update for Microsoft Windows SMB Server (4013389). `https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010`, 2017. [Online; accessed 2-March-2018].

[42] T Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 2013.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[44] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database*. *International journal of lexicography*, 3(4):235–244, 1990.

[45] Sudip Mittal, Prajit Kumar Das, Varish Mulwad, Anupam Joshi, and Tim Finin. Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pages 860–867. IEEE, 2016.

[46] Sagar More, Mark Matthews, Akanksha Joshi, and Tim Finin. A knowledge-based approach to intrusion detection modeling. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 75–81. IEEE, 2012.

[47] Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. *arXiv preprint arXiv:1504.06662*, 2015.

[48] Federico Neri and Paolo Geraci. Mining textual data to boost information access in osint. In *Information Visualisation, 2009 13th International Conference*, pages 427–432. IEEE, 2009.

[49] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. *CoRR*, abs/1510.04935, 2015.

[50] National vulnerability database. http://nvd.nist.gov.

[51] NYTimes. Cyberattack Hits Ukraine Then Spreads Internationally . `https://www.nytimes.com/2017/06/27/technology/ransomware-hackers.html`, 2017. [Online; accessed 2-March-2018].

[52] Onook Oh, Manish Agrawal, and H Raghav Rao. Information control and terrorism: Tracking the mumbai terrorist attack through twitter. *Information Systems Frontiers*, 13(1):33–43, 2011.

[53] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43, 2014.

[54] Line C. Pouchard, Jonathan D. Dobson, and Joseph P. Trien. A framework for the systematic collection of open source intelligence. In *AAAISS*, 2009.

[55] Priyanka Ranade, Sudip Mittal, Anupam Joshi, and Karuna Joshi. Using deep neural networks to translate multi-lingual threat intelligence. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 238–243. IEEE, 2018.

[56] Reddit. https://www.reddit.com/r/cybersecurity/.

[57] The Register. Most vulnerabilities first blabbed about online or on the dark web. `https://www.theregister.co.uk/2017/06/08/vuln_disclosure_lag/`, Jun 2017.

[58] The Register. Make america late again: Us 'lags' china in it security bug reporting. `https://www.theregister.co.uk/2017/10/20/us_china_vuln_reporting/`, Oct 2017.

[59] Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer, 2016.

[60] Anthony Rutkowski, Youki Kadobayashi, Inette Furey, Damir Rajnovic, Robert Martin, Takeshi Takahashi, Craig Schultz, Gavin Reid, Gregg Schudel, Mike Hird, et al. Cybex: the cybersecurity information exchange framework (x. 1500). *ACM SIGCOMM Computer Communication Review*, 40(5):59–64, 2010.

[61] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM, 2010.

[62] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[63] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: state of the art and future challenges. *IEEE Transactions on knowledge and data engineering*, 25(1):158–176, 2013.

[64] Amit Singhal. Introducing the knowledge graph: things, not strings, May 2012.

[65] Nigel C. Smeeton. Early history of the kappa statistic. *Biometrics*, 41(3):795–795, 1985.

[66] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*, pages 926–934, 2013.

[67] Stackoverflow. http://stackoverflow.com/.

[68] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[69] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[70] Zareen Syed, Tim Finin, Ankur Padia, and M. Lisa Mathews. Supporting Situationally Aware Cybersecurity Systems. Technical report, University of Maryland Baltimore County, September 2015.

[71] Zareen Syed, Ankur Padia, M. Lisa Mathews, Tim Finin, and Anupam Joshi. UCO: A unified cybersecurity ontology. In *Proceedings of the AAAI Workshop on Artificial Intelligence for Cyber Security*, pages 14–21. AAAI Press, 2015.

[72] Takeshi Takahashi, Hiroyuki Fujiwara, and Youki Kadobayashi. Building ontology of cybersecurity operational information. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, page 79. ACM, 2010.

[73] Takeshi Takahashi, Youki Kadobayashi, and Hiroyuki Fujiwara. Ontological approach toward cybersecurity in cloud computing. In *Proceedings of the 3rd international conference on Security of information and networks*, pages 100–109. ACM, 2010.

[74] Andranik Tumasjan, Timm Oliver Sprenger, Philipp G. Sandner, and Isabell M. Welpe. Predicting elections with twitter: What 140 characters reveal about political sentiment. In *ICWSM*, 2010.

[75] Peter D Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188, 2010.

[76] Twitter. https://twitter.com/hashtag/cybersecurity?lang=en.

[77] Jeffrey Undercofer, Anupam Joshi, and John Pinkston. Modeling Computer Attacks: An Ontology for Intrusion Detection. In *Proc. 6th Int. Symposium on Recent Advances in Intrusion Detection*. Springer, September 2003.

[78] Alex Vovk. How to Overcome SIEM Limitations. `https://blog.netwrix.com/2016/03/21/how-to-overcome-siem-limitations/`, 2016. [Online; accessed 2-March-2018].

[79] Alex Vovk. Infographics: Common Drawbacks of SIEM Solutions. `https://blog.netwrix.com/2016/03/15/infographics-common-\-drawbacks-of-siem-solutions/`, 2016. [Online; accessed 2-March-2018].

[80] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119. Citeseer, 2014.

[81] Wired. The ransomware meltdown experts warned about is here. `https://www.wired.com/2017/05/ransomware-meltdown-experts-warned/`, 2017. [Online; accessed 2-March-2018].

[82] Peng Xie, Jason H Li, Xinming Ou, Peng Liu, and Renato Levy. Using bayesian networks for cyber security analysis. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 211–220. IEEE, 2010.

[83] Shengqi Yang, Fangqiu Han, Yinghui Wu, and Xifeng Yan. Fast top-k search in knowledge graphs. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 990–1001. IEEE, 2016.

[84] Weiguo Zheng, Lei Zou, Wei Peng, Xifeng Yan, Shaoxu Song, and Dongyan Zhao. Semantic sparql similarity search over rdf knowledge graphs. *Proceedings of the VLDB Endowment*, 9(11):840–851, 2016.