A Knowledge Graph to Represent Software Vulnerabilities

Milad Taghavi

A Thesis

In the Department of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Master of Applied Science (Software Engineering)

at Concordia University

Montreal, Quebec, Canada

February 2023

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Milad Taghavi**

Entitled: A Knowledge Graph to Represent Software Vulnerabilities

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Software Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____Chair
Dr. Joey Paquet

_____Examiner
Dr. Dhrubajyoti Goswami

_____Examiner
Dr. Joey Paquet

_____Supervisor
Dr. Juergen Rilling

_____Supervisor
Dr. Jamal Bentahar

Approved by _____
Dr.

_____
Dr.

Faculty of Engineering and Computer Science

Abstract

A Knowledge Graph to Represent Software Vulnerabilities

Milad Taghavi

Over the past decade, there has been a major shift towards the globalization of the software industry, by allowing code to be shared and reused across project boundaries. This global code reuse can take on various forms, include components or libraries which are publicly available on the Internet. However, this code reuse also comes with new challenges, since not only code but also vulnerabilities these components might be exposed to are shared. The software engineering community has attempted to address this challenge by introducing bug bounty platforms and software vulnerability repositories, to help organizations manage known vulnerabilities in their systems. However, with the ever-increasing number of vulnerabilities and information related to these vulnerabilities, it has become inherently more difficult to synthesize this knowledge. Knowledge Graphs and their supporting technology stack have been promoted as one possible solution to model, integrate, and support interoperability among heterogeneous data sources.

In this thesis, we introduce a methodology that takes advantage of knowledge graphs to integrate resources related to known software vulnerabilities. More specifically, this thesis takes advantage of knowledge graphs to introduce a unified representation that transforms traditional information silos (e.g., VDBs, bug bounty programs) and transforms them in information hubs. Several use cases are presented to illustrate the applicability and flexibility of our modeling approach, demonstrating that the presented knowledge modeling approach can indeed unify heterogeneous vulnerability data sources and enable new types of vulnerability analysis.

Acknowledgements

First and foremost, I am extremely grateful to my supervisor, Dr. Juergen Rilling for his time, effort, and continuous support during my Master study. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly. I would also like to thank Dr. Jamal Bentahar who trusted me in the first place and guided me to a correct path. I must express my gratitude to my sister and brother-in-low, whom without my accomplishment would have not been possible and I was blessed with their support throughout the years.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CAPEC	Common Attack Pattern Enumeration and Classification
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities Exposure
CVSS	Common Vulnerabilities Scoring System
CWE	Common Weakness Enumeration
DL	Description Logic
NVD	National Vulnerability Database
OWL	The Web Ontology Language
RDF	Resource Description Format
SE	Software Engineering
VDB	Vulnerability Database
SW	The Semantic Web
OSS	Open-Source Software

Chapter 1

Introduction

Traditional software development processes focused on closed architectures and platform-dependent software that only support limited code reusability across project and organizational boundaries. With introduction of the Internet, these boundaries have been removed allowing global access, online collaboration, information sharing, and internationalization of the software industry. Software development and maintenance tasks can now be shared amongst team members working inside and outside organizations. Code reuse throughout the different resources such as software libraries, components, services, design patterns, and frameworks published on the Internet has become an essential part of today's software development practice.

Open-Source Software (OSS) publishes source code and other related artifacts using portals such as GitHub[1], or Maven[2]. Such portals allow software artifacts to be shared and reused globally. This reusability can take on different forms, such as integrating open-source projects into existing software ecosystems (e.g., reuse of code libraries) or extending and customizing available projects to meet specific stakeholders needs (e.g., creating specialized Linux distributions). Even though globally shared knowledge resources facilitate and benefit from reuse and collaboration, they also introduce new challenges to the Software Engineering (SE) community. Knowledge

[1] https://github.com/
[2] https://maven.apache.org/

resources are no longer controlled by a single project or organization, instead they are collaboratively managed across projects, organizations, or even global software ecosystems boundaries. Among these challenges, Information Security (IS) has emerged as a major threat to the software development community [1]. The importance of IS for the software community is reflected by the fact that it has become an integrated part of the current SE best practices [2]. At its core, IS promotes the access and use of different security resources during the development process, such as secure coding practices and information about known software vulnerabilities.

In this thesis, we are particularly interested in the integration of information about known software vulnerabilities into the software development process and how they can benefit developers, maintainers, and security experts. Bug bounty platforms [3] along with security vulnerabilities databases (VDBs) [4, 5] are two of the more well-known examples of such knowledge resources, which can provide software professionals with access to existing security issues affecting different software products. Public VDBs (e.g., National Vulnerability Database (NVD[3]) have been introduced to track and publish known software vulnerabilities and solutions to resolve them. These VDBs can be seen as a direct response by the software industry to the ever-increasing number of software attacks, which are no longer limited to a particular project or computer but now often affect hundreds of different systems and millions of computers.

1.1 Motivation

1.1.1 Research Objectives

Weaknesses in software systems are flaws, faults, bugs, vulnerabilities, and other errors that left unaddressed and can make a system vulnerable to attacks. Examples of such flaws include buffer overflows, improper certificate validation, or using components with known vulnerabilities. These software vulnerabilities provide an entry point for attackers, giving affected systems a

[3] https://nvd.nist.gov

greater attack surface [6]. Hackers can leverage this additional attack surface to gain direct access to a system or network. From the moment a vulnerability is discovered until it is patched, a system is potentially exposed to attacks. Since this exposure window leaves systems without protection, it must be as short as possible. This identification, tracking, and reporting of vulnerabilities becomes even more critical in today's software ecosystems, where (vulnerable) components are often shared across projects, organizations, and even software ecosystem boundaries.

To address the removal and management of known security threats, the security community has introduced several knowledge sources that collect information about known software vulnerabilities. For example, VDBs capture information about known software vulnerabilities, their patches, and report on systems affected by these vulnerabilities. However, with this large amount of security vulnerability data being spread across many VDBs, software developers are struggling to identify and locate relevant resources. The situation is further complicated by the fact that individual VDBs are based on different data models. Such heterogeneity in the data representation is often the source of data ambiguity and inconsistency in VDBs and has become another major challenge for organizations managing both disclosure and access to these information resources. In addition, individual VDBs provide access to their information through APIs, RSS feeds, or notification services, while sharing and integration of these resources have remained an open question [7].

There have been many efforts on creating software analytics tools or services to support programmers in managing known vulnerabilities (e.g., [8], [9]). However, the applicability of these approaches is often limited due to several reasons:

1) The proposed tools have remained in information silos by relying on their own proprietary data collection and models. While these models work well for supporting tool-specific analytics services, they limit their ability to share, reuse and integrate data and analysis results with other software analytics tools and knowledge resources.

2) The underlying knowledge models used by current analysis approaches neglect supporting the seamless integration of new knowledge resources or the ability to deal with incomplete data.

3) Their analysis support is often limited by their underlying knowledge models and provides limited flexibility in terms of supporting user-specific analysis needs.

In this thesis, we take the advantage of the Semantic Web (SW) and its technology stack (e.g., ontologies, Linked Data, SW reasoning services) **to establish a unified knowledge graph representation of vulnerability-relevant resources**. Based on this knowledge graph, we can extend and integrate the obtained knowledge with other resources. This enables having a more **flexible and comprehensive global vulnerability analysis** approach that can provide system developers and maintainers with guidance during the management of known vulnerabilities.

The research presented in this thesis is a continuation of work done by Sultan [10] and Eghan [11] on semantic modeling in which they introduced a Security Vulnerability Analysis Framework (SV-AF) that establishes traceability links between NVD, Maven, and the source code of projects. In our research, we extended this work to include additional knowledge resources (e.g., bug bounty programs and additional vulnerability databases) to further enrich the knowledge base that can form the basis for novel vulnerability analysis services.

1.2 Motivating Example

Nowadays many software companies offer solutions that integrate services, components, and code from external resources. Such integrations, provide organizations with added value rather than having to recreate those services or functionalities from scratch. Different techniques (e.g.,

APIs[4], libraries, RESTful[5] interfaces, RPC[6]) are used to build a trustworthy relationship between separate working software applications. While many measures have been taken to maintain software security, applications are still getting penetrated through a trusted third-party system that is vulnerable to some recently exploited weaknesses. Apart from API connections, developers also heavily rely on published libraries by the software community. These external components introduce another penetration port for creating a vulnerable system. While developers can attempt to manually monitor vulnerability databases for known vulnerabilities that might affect their product, this type of monitoring has serval shortcomings.

1.) There are many VDBs and bounty platforms, as well as other resources that may contain information about known vulnerabilities and would require to be monitored to stay ahead of possible weaknesses that might affect their systems.
2.) API dependencies – while a developer might be aware of which APIs or libraries are used in their system, these components might not be directly exposed to vulnerabilities but only indirectly, through the reuse of other external vulnerable libraries. Identifying such transitive dependencies manually is not only difficult but also error-prone since one must consider several factors during this type of analysis, such as library versions and whether (and how) a vulnerable code fragment in these libraries is used.

For example, Log4j[7] is one of the most used logging Frameworks in Java that allows developers to keep track of what happens in their software applications or online services. On December 9, 2021, a Chinese security researcher sounded the alarm about a vulnerability in Log4j[8] and was immediately exploited by hackers. The United States alone was hit with 10 million

[4] https://en.wikipedia.org/wiki/API
[5] https://en.wikipedia.org/wiki/Representational_state_transfer
[6] https://en.wikipedia.org/wiki/Remote_procedure_call
[7] https://logging.apache.org/log4j/2.x/
[8] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228

attempted exploits per hour, with attacks specifically targeting certain critical infrastructure sectors [12]. Companies like Apple, Amazon, Cloudflare, IBM, Microsoft, and Twitter began experiencing a barrage of attacks and many had no choice but to shut down systems until the vulnerability could be resolved [12]. Even the Quebec government shut down nearly 4000 of its sites as a preventive measure [13]. This example illustrates, how a widely used vulnerable source code can impact not only individual applications but also the complete software ecosystem and even commercial systems and organizations are not immune to such vulnerabilities.

Internationally, estimates suggest nearly one-half of global corporate networks experienced a successful exploit in the first five days following the vulnerability's discovery [14]. According to [15], 60% of the Java projects monitored by Snyk that are using the Log4j library are using it as an indirect dependency. Figure 1-1 shows the results of a study conducted in 2020, comparing the direct and indirect vulnerabilities found in 5 popular package manager repositories.

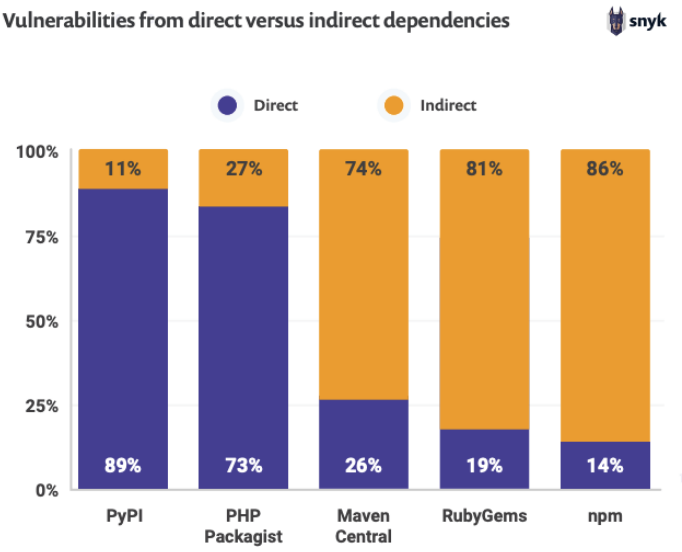

Figure 1-1: Vulnerabilities from direct vs indirect dependencies [15]

The goal of this thesis is therefore to address some of these challenges, by providing a unified knowledge representation of resources and artifacts related to known software vulnerabilities. This unified representation not only allows us to eliminate some of these traditional information silos these resources have remained in but also provides the foundation for novel

software analytics services that can support impact and ripple effect analysis of vulnerabilities on software products and even complete software ecosystems.

1.3 Contributions

The main contributions of our work are as follows:

- A review of known software vulnerability resources (Chapter 2)
- **Proposing a methodology** that establishes a knowledge graph for software artifacts. (Chapter 3)
- Introduce a knowledge graph for software vulnerability related resources based on our knowledge modeling methodology. (Chapter 4)
- Discuss major design decisions we applied to enhance the ontology design and illustrate how **our approach takes advantage of reasoning services**. (Chapter 4)
- Illustrate how our integrated knowledge modeling approach can support **novel types of knowledge-driven software analytics services.** More specifically, we show examples of vulnerability analysis across VDBs and Bug Bounty Platforms. (Chapter 5)

The remainder of the thesis is organized as follows: Chapter 2 provides background related to bug bounty programs and semantic web technologies. Chapter 3 presents a methodology to create a knowledge graph for software artifacts. Chapter 4 applies our methodology to create a knowledge graph for known software vulnerabilities. Chapter 5 demonstrates some use cases using developed ontology and acquired data. Related work and discussions are discussed in Chapter 6. Finally, Chapter 7 offers concluding remarks and outlines future research directions you may find Appendices in Section 8 for more information

Chapter 2

Background

In this chapter, we provide a brief overview of the core techniques and terminologies used in our research. If you are already familiar with these concepts, you can safely move on to the next chapter as cross-references are provided wherever specific background information is required.

2.1 The Semantic Web

The Semantic Web has been defined by Berners-Lee et al. as "an extension of the Web, in which information is given well-defined meaning, better-enabling computers and people to work in cooperation" [16]. It forms a Web from documents to data, where data should be accessed using the general Web architecture (e.g., URIs). Using this Semantic Web infrastructure allows data to be linked, just as documents (or portions of documents) are already, allowing data to be shared and reused across application, enterprise, and community boundaries. Some of the knowledge modeling challenges which are addressed by the Semantic Web include vastness, vagueness, uncertainty, inconsistency, and deceit of information. Ontologies are an important foundation of the SW as they allow knowledge to be shared between different agents and the creation of common terminologies for understanding [17]. In computer science, a widely accepted definition has been introduced by Studer [18]: "an ontology is a formal, explicit specification of a shared conceptualization." Ontologies allow the specification of concepts and relationships in a domain of discourse. Having such a formal representation improves portability, flexibility, and information-sharing problems associated with traditional databases [19]. In addition, while

traditional database systems assume complete knowledge (closed world assumption), ontologies support the modeling of incomplete knowledge (open world assumption), as well as the extensibility of the knowledge model (ontologies).

In a Semantic Web, data can be processed by computers as well as by humans, including inferring new relationships among pieces of data. For machines to understand and reason about knowledge, this knowledge needs to be represented in a well-defined, machine-readable language. The Semantic Web can be considered a collaborative project that is planned and designed to achieve a universal medium for exchanging information on the World Wide Web in a way that machines can understand and process this information. To provide this meaningful information for machines, there is a need in creating data that describes data on the web which is called Metadata. Consequently, Metadata gives machines a method to process the meaning of things which is called semantics. Hence, when computers have semantics, they are qualified for solving complex semantical optimization problems such as returning relevant search results.

2.2 The Semantic Web and its Technology Stack

Ontologies are an important foundation of the SW, as they allow knowledge to be shared between different agents and the creation of common terminologies for understanding [17]. The current data model used to represent this meta-data in SW is the Resource Description Format (RDF). RDF is used to formalize meta-models in the form of <subject, predicate, object>, which are called triples. RDF triples make statements about resources, with a resource in the SW being anything—a person, project, software, a security bug, etc. To make triples persistent, RDF triples stores are used, with each triple being identified by a Uniform Resource Identifier (URI). The Web Ontology Language (OWL) [20] is used on top of the RDF layer (see Figure 2-1). It is a standard modeling language put forward by the W3C to pursue the vision of the SW. OWL provides for machine understandable (i.e., capturing semantics) information, allowing Web resources to be automatically processed and integrated. The widely used OWL sub-language OWL-DL is based on the Description Logics (DLs) [21].

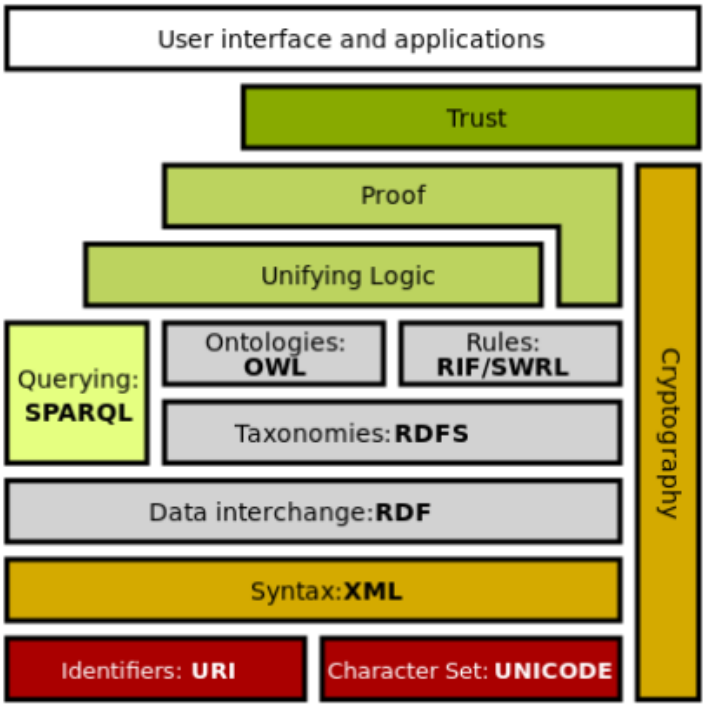

Figure 2-1: Semantic Web technology stack. [22]

2.2.1 Description Logic

A DL-based knowledge representation system provides typical facilities to set up knowledge bases and to reason about their content [21]. Figure 2-2 illustrates a typical DL-based knowledge system. Such a knowledge base (KB) consists of two components—the TBox contains the terminology (i.e., the vocabulary of an application domain), and the ABox contains assertions about named individuals in terms of this vocabulary. The terminology is specified using description languages introduced previously in this section, as well as terminological axioms, which make statements about how concepts or roles are related to each other.

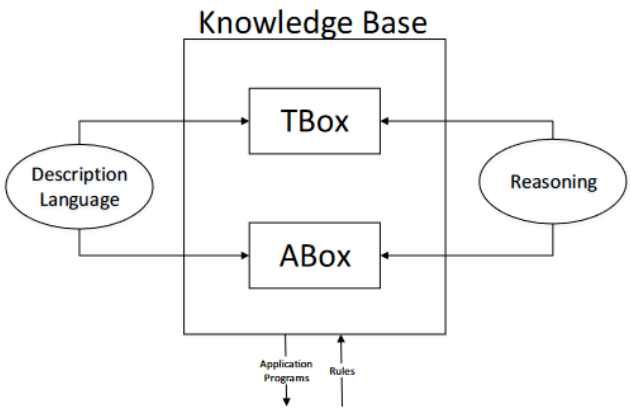

Figure 2-2: Description Logic System

In the most general case, terminological axioms have the form:

$$C \sqsubseteq D \ (R \sqsubseteq S) \ or \ C \equiv D \ (R \equiv S),$$

where C and D are concepts (R and S are roles). The semantics of axioms are defined as: an interpretation $\boldsymbol{I}$ satisfies $C \sqsubseteq D$ ($R \sqsubseteq S$) if $C\boldsymbol{I} \sqsubseteq D\boldsymbol{I}$ ($R\boldsymbol{I} \sqsubseteq S\boldsymbol{I}$).

A TBox, denoted as $\boldsymbol{T}$, is a finite set of such axioms. The assertions in an ABox are specified using concept assertions and role assertions, which have the form $C(a)$, $R(a, b)$, where C is a concept, R is a role, and a, b are names of individuals. A DL system not only stores terminologies and assertions but also offers services that allow reasoning about them. Typical reasoning services for a TBox are to determine if a concept is more general than another (i.e., subsumption) or if a concept is satisfiable (i.e., non-contradictory). Reasoning services for an ABox are to find out whether its set of assertions is consistent, and whether the assertions in an ABox entail that a particular individual is an instance of a given concept description.

A DL knowledge base might be embedded into an application in which some components interact with the KB by querying and modifying the knowledge, i.e., by adding and retracting concepts, roles, and assertions. However, many DL systems also provide an application

programming interface that consists of functions with well-defined logical semantics, which application programs can use to operate on the KB [21].

2.2.2 The Resource Description Framework (RDF)

RDF was developed by the World Wide Web Consortium (W3C) to allow for the automated semantic processing of information, by structuring information using individual statements that consist of (Subject, Predicate, Object). RDF is the foundation of the Semantic Web and what provides its innate flexibility. All data in the Semantic Web is represented in RDF, including schema describing RDF data. Although frequently referred to as a 'language', RDF is mainly a data model. It is based on the idea that the things being described have properties, which have values, and that resources can be described by making statements. RDF prescribes how to make statements about resources, in particular, Web resources, in the form of subject-predicate-object expressions. RDF is not like the tabular data model of relational databases. Nor is it like the trees of the XML world. Instead, RDF is a graph and is classified as a No-SQL database type.

The basic RDF components include statements, Uniform Resource Identifiers, properties, blank nodes, and literals. RDF-star (formerly RDF*) extends RDF with support for embedded triples. RDF is a bunch of nodes connected by edges where both the nodes and edges have labels.

Web Ontology Language (OWL)

OWL is an ontology language for the Web. OWL is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. OWL is a computational logic-based language such that knowledge expressed in OWL can be exploited by computer programs, e.g., to verify the consistency of that knowledge or to make implicit knowledge explicit. OWL documents, known as ontologies, can be published on the World Wide Web and may refer to or be referred to from other OWL ontologies. OWL is part of the W3C's Semantic Web technology stack, which includes RDF, RDFS, SPARQL, etc. The current version of OWL, also referred to as "OWL 2", was developed by the W3C OWL Working Group and published in 2009, with a Second Edition published in 2012. Syntactically, an OWL

ontology is a valid RDF document and as such also a well-formed XML document. This allows OWL to be processed by the wide range of XML and RDF tools already available.

Its primary uses are fast and flexible data modeling and efficient automated reasoning. Semantically, OWL is based on the description logics [21]. Generally, description logics are a family of logics that are decidable fragments of first-order predicate logic. These logics focus on describing classes and roles and have set-theoretic semantics.

2.2.3 SPARQL

SPARQL is a query language for dealing with information from RDF graphs. As a data access language, it is suitable for both local and remote use. It provides facilities to:

- extract information in the form of URIs, blank nodes, and literals.
- extract RDF subgraphs.
- construct new RDF graphs based on information in the queried graphs.

The RDF data model expresses information as graphs, consisting of triples with subject, predicate, and object. Many RDF data stores hold multiple RDF graphs, and record information about each graph, allowing an application to make queries that involve information from more than one graph. The SPARQL query language is based on matching graph patterns. A SPARQL query is executed against an RDF Dataset which represents such a collection of graphs. Different parts of the query may be matched against different graphs. There is one graph, the default graph, which does not have a name, and zero or more named graphs, each identified by IRI [23].

There are three types of data sources that can be queried: RDF files, Triple Stores, and SPARQL endpoints.

- *RDF file* – A file containing RDF can be loaded by an application into its corresponding graph and be queried against. RDF can be held in several different file formats (i.e. .rdf, .xml, .nt, .owl, .ttl).

- *Triple Store* – A specialized database used for the storing and retrieving of RDF statements. Every record in the database must follow the triple format of subject-predicate-object. A triple store can also be referred to as an 'RDF Data Store' as well as an 'RDF Database' [24].
- *SPARQL Endpoint*[9] – A web service interface for human or machine users to provide SPARQL queries and receive results. SPARQL endpoints are important for Linked Data since they allow datasets to be publicly accessible.

In addition to RDF, OWL, and OWL-DL, the SW community provides tools to process OWL semantics and RDF data. Jena[10] emerged as a Java framework for building applications and providing a programmatic environment for RDF and OWL. Reasoners (e.g., Jena) can infer new facts about the designed ontology and form a set of asserted axioms. RDF databases, such as Virtuoso[11] and AllegroGraph[12], are used to materialize and store RDF triples. SPARQL is an RDF query language, that is, a semantic query language for databases able to retrieve and manipulate data stored in RDF format.

2.3 Knowledge Graphs

A graph is one of the fundamental data abstractions in computer science. Virtually every graph application needs to store and query the graph. The knowledge graph is a virtual data layer on top of the existing databases or data sets to connect all data—whether structured or unstructured—at scale [25]. KGs contain a large amount of prior knowledge but can also effectively organize data. Primarily, a knowledge graph represents, among other things, a model of a knowledge domain created by experts using intelligent algorithms in the machine learning [25]. A knowledge graph is a structured representation of facts, consisting of entities, relationships,

[9] https://www.w3.org/wiki/SparqlEndpoints
[10] https://jena.apache.org/
[11] https://virtuoso.openlinksw.com/
[12] https://allegrograph.com/

and semantic descriptions. Entities can be real-world objects and abstract concepts, relationships represent the relation between entities, and semantic descriptions of entities, and their relationships contain types and properties with a well-defined meaning. They have been widely used for question-answering systems, search engines, and recommendation systems. All these facets rely on the support of knowledge reasoning over knowledge graphs. Knowledge reasoning over knowledge graphs aims to identify errors and infer new conclusions from existing data. New relations among entities can be derived through knowledge reasoning and can feed back to enrich the knowledge graphs, and then support the advanced applications. Considering the wide application foreground of knowledge graphs, the study of knowledge reasoning on large-scale knowledge graphs has become one research focus in natural language processing in the past few years [26].

The term knowledge graph is often used as synonymous with knowledgebase however, there is a difference. Knowledge graphs can be viewed as a graph due to their graph structure. While it involves formal semantics, it also serves as a knowledge base for interpretation and inference over facts [27, 28]. Knowledge-aware models benefit from the integration of heterogeneous information, rich ontologies and semantics for knowledge representation, and multi-lingual knowledge. Ontologies represent the backbone of the formal semantics of a knowledge graph. They can be seen as the data schema of the graph and serve as a formal contract between the developers of the knowledge graph and its users regarding the meaning of the data in it.

2.3.1 Knowledge Graph vs Linked Data

A key feature of a KG is that entity descriptions should be interlinked with one another. The definition of one entity includes another entity. This linking is how the graph forms, (e.g. A is B. B is C. C has D. A has D). Knowledge bases without formal structure and semantics, (e.g. Q&A knowledge bases) do not represent a KG.

Linked Data by definition is structured data that is interlinked with other data. It builds upon a suite of web standards such as HTTP, RDF, and IRIs, to allow the linking of data, so that

a person or machine can explore this web of data [29]. In [30] linked data was defined as "a set of best practices for publishing and connecting structured data on the Web.". It focuses on interconnecting data and resources on the Web by defining relations between ontologies, schemas, and/or directly linking the published data to other existing resources on the Web. Linked data enables publishing machine-readable interpretation of heterogeneous sources of information. It is possible to have an expert system that has a collection of data organized in a format that is not a graph but they have to rely on automated deductive processes such as a set of 'if-then' rules to facilitate analysis.

Knowledge Graphs are created by describing Entities and Entity Relationships that are deployed using Linked Data principles. Knowledge graphs acquire and integrate information into an ontology and reasoner can then be used to derive new knowledge [31]. Linked data is generally considered fundamental to building a knowledge graph, linked data is not in itself a knowledge graph. In other words, linked data is a necessary but not sufficient condition for knowledge graph construction.

2.3.2 Examples of Big Knowledge Graphs

DBpedia is a crowd-sourced community effort to extract structured content from the information created in various Wikimedia projects[13]. This structured information resembles an open knowledge graph (OKG) which is available to everyone on the Web. DBpedia data served as Linked Data, which is revolutionizing the way applications interact with the Web. One can navigate this Web of facts with standard Web browsers, and automated crawlers or pose complex queries with SQL-like query languages (e.g., SPARQL). DBpedia is connected with other Linked Datasets through approx. 62 million RDF links. As of June 2021, The DBpedia core KG contains more than 850 million triples [32].

[13] https://www.dbpedia.org/about/

Another big KG is the Google Knowledge Graph [33], which enables search for information published on the Internet. Google's Knowledge Graph is not just rooted in public sources such as Freebase, Wikipedia, and the CIA World Factbook, it is also augmented at a much larger scale and currently contains more than 500 million objects, as well as more than 3.5 billion facts about these objects and their relationships. The KG design is optimized to support user searches and the resources available on the Web [33]. Figure 2-3 is an illustration of mapped concepts in DBpedia.

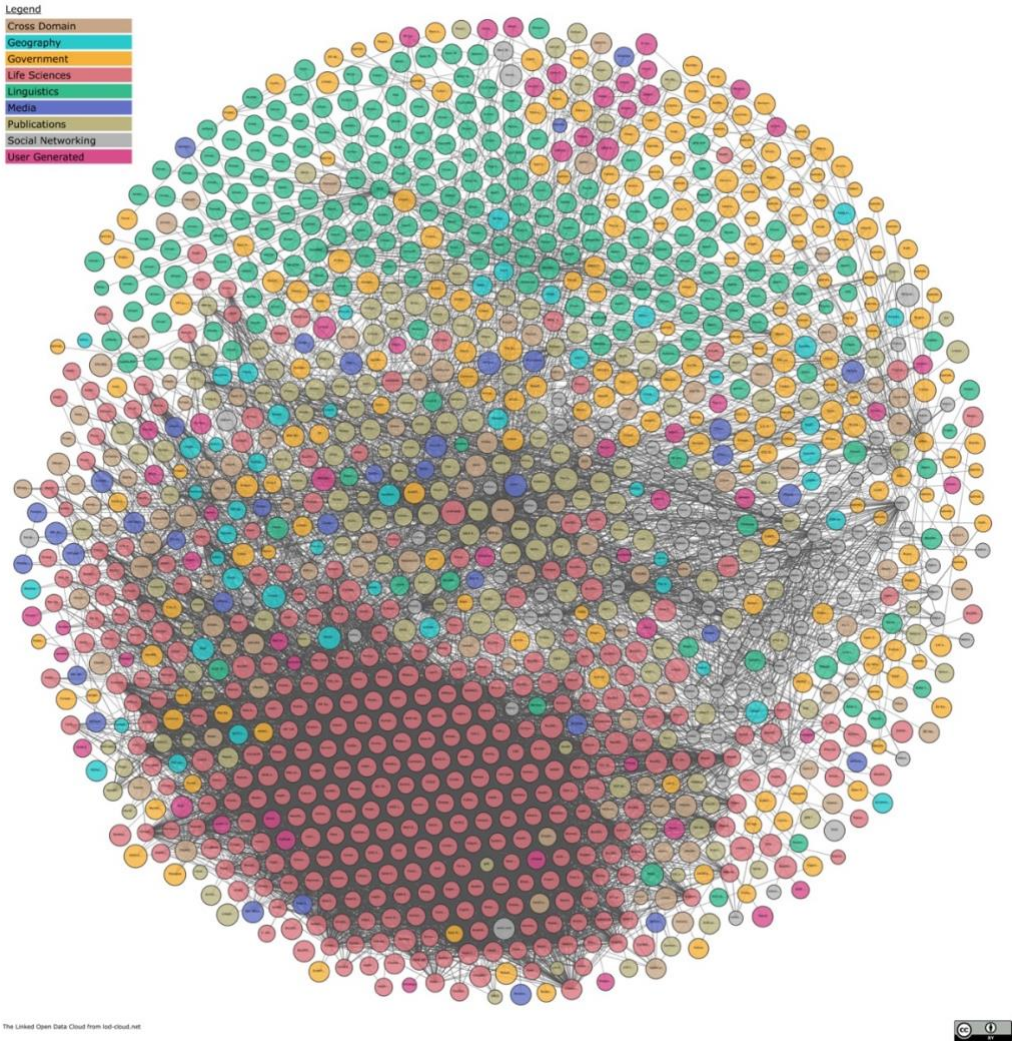

Figure 2-3: DBpedia concepts map [34]

2.3.3 Graph Databases versus Relational Databases

The following section compares briefly graph databases used by the Semantic Web with traditional relational databases. Relational databases exist for many decades and are the database technology of choice for most traditional data-intensive storage and retrieval applications. In [35] authors presented analytical research and compared relational databases with graph databases. The paper found that value retrieval or data processing is very fast in graph databases compared to RDBMS. They identified that a graph database performs much better than RDBMS when it comes to objectivity, also graph databases excel when it comes to data integration. Relational databases are typically closed systems. One approach to data integration relies on creating a global schema that captures the interrelationships between the data items represented across these databases. However, creating such a global schema is extremely difficult, since schemata and the meaning of the data they store will vary from database to database. Another problem with creating such a global schema would be its extendibility and evolution in terms of adding new resources (databases) or modifying existing schemata. These changes might potentially impact different parts of the global schema and might require significant changes to applications using this global schema. Because of the challenges in creating a global schema, it is convenient to sidestep this issue and convert the relational data into a database with the generic schema of triples, i.e., a knowledge graph. The mappings between the attributes are created on an as-needed basis, for example, in response to addressing specific business questions, and can themselves be represented within a knowledge graph.

2.4 Vulnerability Databases and Repositories

To date, software developers have often overlooked security issues throughout the software development lifecycle. Existing software design and engineering processes provide little guidance about security, and the communication disconnect between software developers and cybersecurity experts has led to the widespread introduction of software vulnerabilities [36]. This lack of

awareness often results in: 1.) exploitable vulnerabilities in software systems and 2.) developers being unaware that their software might have been exposed to a vulnerability.

The situation is further complicated, with today's software system depending more and more often on external components and libraries, which are managed by external stakeholders. Requirements engineers have long recognized the importance of requirements dependency analysis to discover and manage critical relationships among requirements [37]. According to [37], up to 70% of total software errors are caused by interacting requirements, making requirements dependency errors a significant software development and quality challenge. For example, in 2018, a data breach caused by a security vulnerability exposed more than 50 million Facebook user accounts to malicious attackers [38]. This data breach was caused by two requirements, namely "view as" and "upload birthday video" [39]. Although "upload birthday video" was introduced in 2017, its interdependency with other requirements (especially with "view as") was not thoroughly tested, resulting in a serious security breach.

VDBs are the result of an effort by the security community to collect information about all known security flaws in software. A vulnerability database is a platform for collecting, maintaining, and disseminating information about discovered computer security vulnerabilities. From the outset, this has been a massive challenge because vulnerability information is generated by thousands of sources including software vendors, vulnerability researchers, and software users. A VDB describes the identified vulnerability, assesses the potential impact on affected systems, and any workarounds or updates to mitigate the issue. VDBs assign a unique identifier to each vulnerability cataloged such as a number or alphanumeric designation and make the vulnerability information available via web pages, exports, or API. In Table 2-1 we listed some of the well-known VDBs and the features they provide.

Database	Established	Public	API	Vulnerability
CVE[14]	1999	Yes	No	150,799
NVD[15]	2005	Yes	Yes	159,865
Vuldb[16]	1997	Yes	Yes	171,574
SecurityFocus Vulnerability DB[17]	1999	Yes	No	From CVE
SecurityTracker[18]	2001	No	No	From CVE
ZeroDayInitiative[19]	2005	Yes	Yes	From CVE
Exploit Database[20]	2010	Yes	No	From CVE
Japan Vulnerability Notes[21]	2007	Yes	No	From CVE
AusCERT Bulletins[22]	1993	Yes	Yes	From CVE
CERT-EU Security Advisors[23]	2012	Yes	No	From CVE
VulnBD Cyber risk[24]	2011	No	Yes	250,756

Table 2-1: List of some of the known VDBs

NVD is one of the most widely used vulnerability databases. It was established in 2005 to provide a U.S. government repository for data about software vulnerabilities and configuration settings. The objective of NVD was to use open standards to provide reliable and interoperable information about vulnerabilities, their impact metrics, technical assessment methods, IT product identification data, and references to the remediation assistance [4]. NVD is maintained by the US government, strives to accurately document all publicly known vulnerabilities, and effectively serves as the industry's standard. Both commercial security services, and open-source security tools depend on the NVD's vulnerability information to function effectively [40].

[14] https://cve.mitre.org/
[15] https://nvd.nist.gov/
[16] https://vuldb.com/
[17] https://bugtraq.securityfocus.com/
[18] http://www.securitytracker.com/
[19] https://www.zerodayinitiative.com/
[20] https://www.exploit-db.com/
[21] https://jvn.jp/en/
[22] https://www.auscert.org.au/bulletins/
[23] https://cert.europa.eu/
[24] https://vulndb.cyberriskanalytics.com/

NVD is based on the list of Common Vulnerability and Exposures (CVE) entries. Using CVEs and CVE identifiers ensures that unique vulnerabilities are discussed, and that information about the same vulnerability is shared by different parties. NVD is often interchangeably used with the CVE list but there are some differences between the two resources despite having a very close relationship.

The CVE dictionary was launched in 1999, five years before the NVD, and is run by the non-profit MITRE Corporation which was mentioned above. Whereas the NVD is a more robust dataset describing the vulnerabilities, the CVE dictionary is more barebones, providing the straight facts of the CVE ID number (CVE-year-unique id #), as well as one public link. To put it simply, the CVE is the organization that receives submissions and IDs them, while the NVD adds the analysis and makes it easier to search and manage them.

CWE seeks to make vulnerability management more streamlined and accessible. The community-developed catalog features hardware and software weaknesses and is described as a common language, to be used by security tools, and as a baseline for weakness identification, mitigation, and prevention efforts. CWE can be considered a dictionary of software vulnerabilities, while CVE is a list of known instances of vulnerability for specific products or systems. NVD integrates CWE into the scoring of CVE vulnerabilities by providing a cross-section of the overall CWE structure.

While NVD is the largest and most well-known VDB, is not the only one. The CERT Vulnerability Notes Database, includes summaries, technical details, remediation information, and lists of affected vendors. Most vulnerability notes are the result of private coordination and disclosure efforts. VulnDB is a commercial Risk Based Security vulnerability database. VulnDB tracks vulnerabilities in third-party libraries and claims to cover over 47,000 vulnerabilities that are not found in CVE or NVD. SecurityTracker is another commercial vulnerability dataset that covers vulnerability entries that have CVE-IDs. Another publicly available vulnerability database

is exploit-db. The Exploit Database[25] is a CVE-compliant archive of public exploits and corresponding vulnerable software, developed for use by penetration testers and vulnerability researchers.

2.5 Bug Bounty Programs

Whether it is a small or a large organization, internal security teams require an external audit from other real-world hackers to test their applications for them. An increasingly popular approach to identifying vulnerabilities in software is to offer rewards to security researchers that are external to an organization ("hackers") to find and disclose vulnerabilities [3]. Such bug bounty programs, also known as vulnerability rewards programs are crowd-sourced mechanisms that allow companies to pay hackers individually for their work in identifying vulnerabilities in their software. Application vendors pay (bounty) hackers to detect and identify vulnerabilities in their software, web applications, and mobile applications. These bounty hunters produce vulnerability reports and send them to the company that owns the program to fix those flaws quickly. If the report is accepted by the company, the reporter gets paid [41].

Public bug bounty programs

These programs are open to anyone who wants to participate. This program may prohibit some participants based on their reputation and track record. In general, anyone can participate in a public bounty program, which specifies the scope, the rules of engagement, as well as the bounty guidelines.

[25] https://www.exploit-db.com

Private bug bounty programs

Invite-only programs are available for selected participants, often selected based on their skill level, experience with the particular application, and other statistics.

There are a few differences between a public and private program. Conventionally, programs tend to start as private and over time evolve into public programs. This is not always true but, mostly, businesses start a private bug bounty program and invite a group of researchers or developers to test their applications before the program goes public to the community. Often, companies will not open their programs to the public, to allow them to control the hacker activity in the sections that are critical to the organization. This reduces the number of low-severity vulnerabilities in out-of-scope applications. Many organizations use also this approach to verify their security posture.

2.5.1 Bug Bounty platforms

Bug bounty platforms manage bounty programs for different companies, including advertisement and matching of new bounties with hackers/bounty hunters, and managing the reward payments. Some of the most popular public platforms are listed below.

HackerOne

HackerOne[26] is one of the major vulnerability collaboration and bug bounty hunting platforms that connect companies with bounty hunters. It was one of the first start-ups to commercialize and utilize crowd-sourced security and hackers as a part of its business model. By May 2020, HackerOne has made a total of $100 million in the bug bounty payments [42], with half of HackerOne's bounties being paid just in 2019, and Gartner projects that by 2022, 50% of enterprises will employ crowdsourced cybersecurity [43].

[26] https://hackerone.com/

Bugcrowd

Bugcrowd[27] Inc. is another leading in the crowd-sourced cybersecurity field. The company has developed a coordination platform that connects businesses with security researchers to test their applications [43].

Synack

Synack[28] includes a vulnerability intelligence platform that automates the discovery of exploitable vulnerabilities for reconnaissance and turns them over to the company's freelance hackers to create vulnerability reports for clients. Synack is used by the top 25 enterprise software start-ups [44]

[27] https://www.bugcrowd.com/
[28] https://www.synack.com/

Chapter 3

A Methodology to Establish a Knowledge Graph for Software Artifacts

Software development involves software artifacts that are not limited to source code (e.g., build scripts, documentation, issue trackers), created by different stakeholders and used during different phases of the development process. The main issue is that these artifacts have remained in information silos with limited support for knowledge sharing and reuse across artifact boundaries. The Mining software repository community has addressed this problem by making historical data found in these knowledge resources not only actionable data, but also improving the accessibility and traceability of these resources. Various techniques have been applied, such as the machine learning [45, 46], information retrieval [47], and summarization techniques [48] to assist development teams during software analytics tasks including code search, duplicate bug report detection, traceability link recovery [49].

Another more recent research avenue to link software artifacts is knowledge graphs that are used to model software artifacts. In [50] a methodology is introduced to locate source files, given a natural language description of bugs that uses a knowledge graph. Another approach was extracting and indexing artifacts to form a knowledge graph. Also, [51] proposed to construct a knowledge graph of a code repository to create links between issue reports and corresponding code commits.

In this research, we build upon this existing research and introduce a new methodology that takes advantage of knowledge graphs to provide a standardized knowledge representation to support the linking and inference of new knowledge across software artifact boundaries. More specifically, our methodology integrates knowledge resources found in the software vulnerability domain, such as VDBs and bounty programs. The goal of our proposed methodology is to provide a step-by-step process of constructing such a knowledge graph.

Developing such a vulnerability knowledge graph is an iterative process since the ontologies used for the data representation have to be sufficiently expressive and flexible to allow knowledge reuse and sharing, as well as for the inference of new knowledge to support different SE tasks. In this chapter, we walk through the basic steps of our methodology, as illustrated in Figure 3-1. These steps include the selection and acquisition of knowledge resources (Section 3.1), knowledge modeling process (Section 3.2), data extraction (Section 3.3), data cleaning and preprocessing (Section 3.4), the population of the knowledge graph (Section 3.5) and how this unified knowledge model can be used to support various software vulnerability related software analytics tasks (Section 3.6). Also, the evolution of the knowledge model will be addressed (Section 3.7)

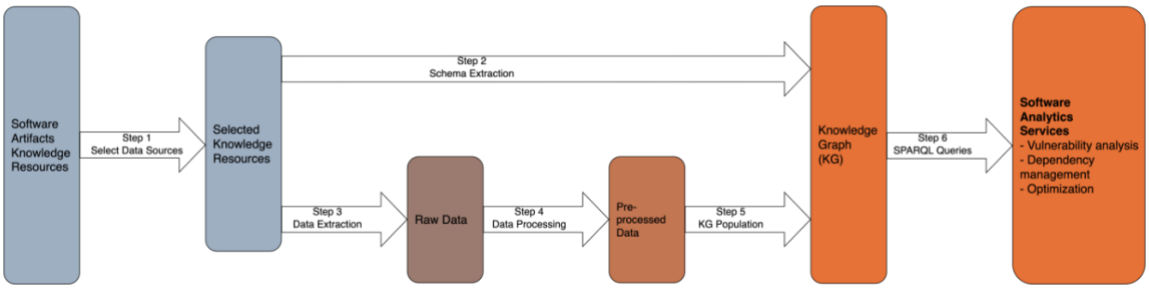

Figure 3-1: KG development process

3.1 Selection and acquisition of knowledge resources (Step #1)

As discussed throughout the thesis, vulnerabilities and weaknesses in software systems are one of the primary causes of security threats and breaches. These vulnerabilities not only affect the usability of these affected systems, but software productivity and competitiveness are also increasingly dependent on the successful management of vulnerability information. As discussed earlier (Chapter 2), various resources exist to manage and centralize vulnerability-related information that are presented in Figure 3-2.

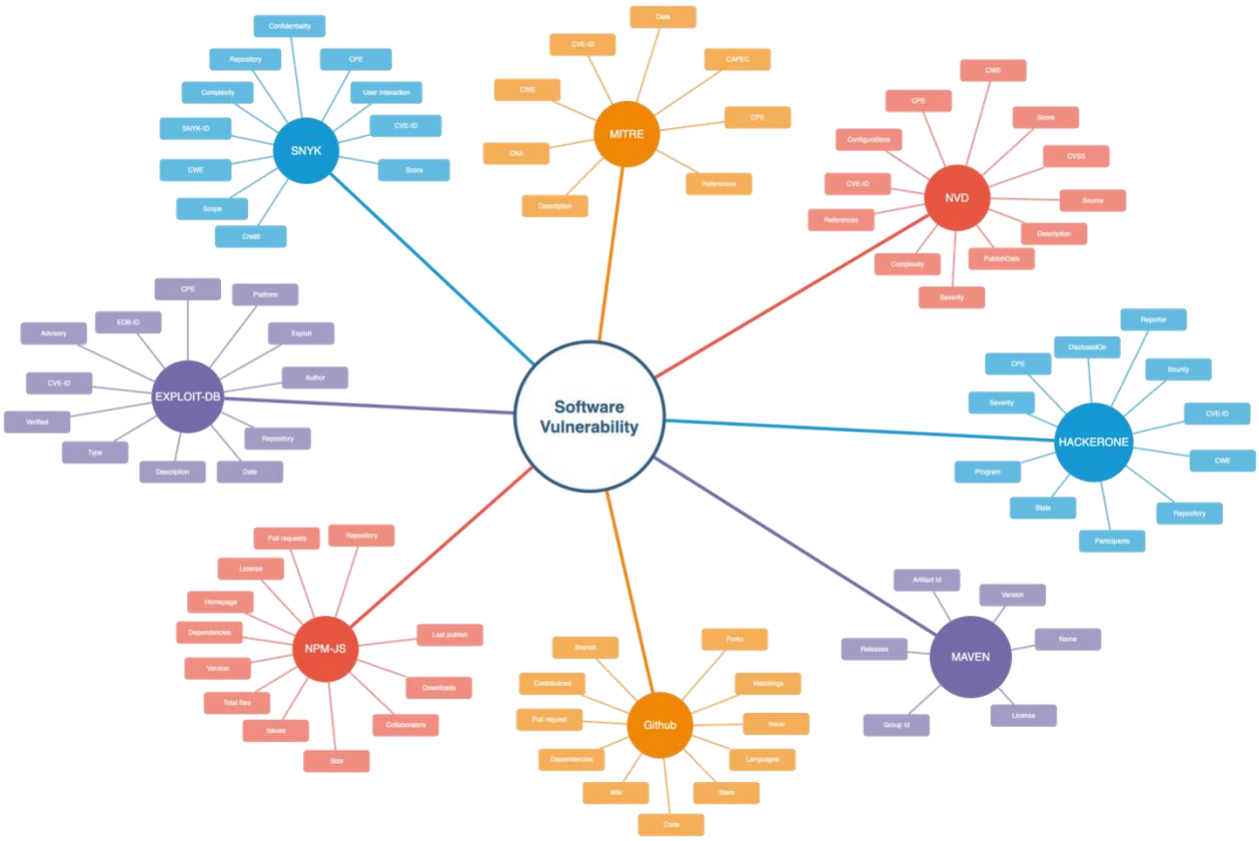

Figure 3-2: Schematic KG view

The key to any successful knowledge modeling approach is to (1) identify knowledge resources that are *rich* enough to provide meaningful information for specific application contexts, that are (2) *reliable* in terms of the quality of the information they provide, and they are (3) readily *accessible*. Here we explain the key factors for our purposes.

Richness: We define a knowledge resource to be rich enough if it contains or references information related to known vulnerabilities, information that can be used to classify vulnerabilities or it can be used to capture potential effects of known vulnerabilities on software systems.

Reliability: For the reliability of an information resource, we only consider resources for which the content is actively managed. For example, resources that are managed as part of a community effort (e.g., VulDB), by the government (e.g., NVD), an organization (e.g., HackerOne), or a combination of these. Such information management should include but is not limited to the validation of information stored in the information resource, regularly updated, information consistency (e.g., removal of outdated, no longer valid information).

Accessibility: We only consider data sources that are readily (publicly) accessible either through a Web interface/API, as a dataset in a standard representation format (e.g., JSON), or which can be extracted (crawled) using existing libraries (e.g., libraries.io).

Ontologies can capture highly complex ideas and business logic by providing not only a standardized approach for representing knowledge, but also supporting the inferences of new knowledge using SW reasoners to infer symmetry, asymmetry, inversion, composition, and transitivity relations. Any ontology design effort should be based on an incremental modeling approach, where one should identify a few critical questions that the ontology needs to answer. Modeling for only a few use cases will provide immediate value to the end-users (by having a working model implemented quickly) and allows optimizing the ontology for these specific use cases. This is a contradictory approach with modeling a full domain which typically leads to an ontology design that might be too complex and not optimized for knowledge interference and linking of sub-ontologies.

We identify the following initial four use cases which our knowledge graph should support:

1. Scanning for vulnerabilities in a project or any of its direct or indirect dependencies
2. Measuring the number of projects potentially impacted by a bounty hunter with considering the number of dependents each project has
3. Combining HackerOne reports with CWE records to find areas of expertise for a bounty hunter by counting the number of CWE-IDs
4. Being able to integrate data from this KG to other remote graphs

Knowledge graph provides us with the ability to facilitate knowledge inferences such as symmetry, asymmetry, inversion, composition, and transitivity relations. We then further refine and enrich the initial design of these ontologies by adding additional relations and properties. This enables us to have a semantic model that is rich enough to allow the inference of knowledge using basic SW reasoning (RDFS++), which includes inferences such as sameAs inverseOf, TransitiveProperty. Protégé[29] is the most well-known ontology editor, developed by Stanford University in the 1980s [52]. Protégé supports features of OWL and RDFS and also includes various reasoners to test and query the ontology. After modeling the ontology in Protégé is completed, the ontology will be imported to our triple-store.

3.2 Schema Extraction and Ontology Design (Step #2)

Given that ontologies have been used for years, many ontologies have already been created to model various domains. Reusing an existing ontology will not only reduce the time it takes to design an ontology but also contributes towards the standardization and reuse of knowledge models. The Linked Open Vocabularies[30] portal, provides a collection of open-source vocabularies and ontologies such as DBpedia[31], Friend of Friend, or Schema.org [53]. These general ontologies

[29] https://protege.stanford.edu/
[30] https://lov.linkeddata.es/dataset/lov/
[31] https://www.dbpedia.org

can be used to extract alternative descriptions and labels for defining classes and relationships. Our knowledge modeling process consists of three major steps and is inspired by the methodology introduced by Noy et al. [54], as well as a bottom-up knowledge modeling approach similar to the one used by Van der Vet et al. [55]. We first perform a *manual review* of the documentation from selected vulnerability-relevant software artifacts and existing ontologies to identify and extract concepts and properties used by the individual resource. Next, we *manually inspect* these extracted concepts and properties for a resource to derive an initial version of the corresponding system-specific ontologies. These system-level ontologies are not yet optimized in terms of knowledge reuse, integration, or inference of new knowledge. For the model optimizations, we apply an iterative bottom-*up* modeling approach. During this bottom-up modeling approach, we identify and extract shared concepts and attributes from these system-specific ontologies into different layers of abstraction (upper ontologies). During this ontology design step, we also consider our different use cases and introduce additional relations, constraints, and properties to be able to take advantage of semantic web reasoning services.

3.3 Data Extraction (Step #3)

After establishing the knowledge graph structure, data (facts) must be extracted from the original knowledge resources. Data extraction is therefore the first step in a data ingestion process called ETL (extract, transform, and load). The goal of ETL is to prepare data for analysis. The facts are extracted from (public) repositories or SaaS (Software as a Service) platforms. Since we consider numerous artifacts that contain data at various abstraction and representation types, different data extraction methodologies are required. As part of the extraction process, data has to be extracted from files, web services, or using scraping scripts directly from web pages. While many of the data sources (e.g., VDBs) are already stored in a structured and machine-readable format, other resources will require additional techniques (e.g., Information Retrieval [56]) to extract the facts from their unstructured representations. As part of the extraction process, one also must consider how the facts are updated by the various knowledge providers to ensure the long-

term relevance and consistency of the extracted data. In general, there are three primary approaches provided by the knowledge resources:

- **Full extraction:** Knowledge resources provide regular complete data dumps. These snapshots/data dumps contain the full data history. However, they will require additional post-processing to extract recent updates or to repopulate the knowledge graph (after performing a full data cleaning on the often very large data dump) with all the facts.
- **Incremental extraction:** Some data sources do provide regular incremental updates to their knowledge base. During the extraction process, one must identify and propagate changes. One drawback of incremental extraction is that it may not be able to detect deleted records in the source data. Another drawback is that the data is updated on-demand only (no real-time data synchronization).
- **Update notifications:** The ability to subscribe to knowledge changes and to be notified by the knowledge resources if new facts/data become available. All we need to do is to handle published data updates. This data extraction approach has several advantages in terms of keeping the modeled knowledge resources up-to-date. Unfortunately, many of the knowledge sources do not yet provide this feature.

3.4 Data Pre-processing (Step #4)

After the fact extraction, some data pre-processing and data cleaning will be required to improve data quality. Data quality is affected by data inconsistencies, data duplication as well as general noise in the data (non-relevant or even wrong data) that needs to be dealt with during the data processing step. If data quality is low, later analysis of this data will be affected. Data cleaning is a process of polishing the data and removing noise in a dataset. While data cleaning is not a standardized process and depends on the data source, the following processing steps are generally performed:

A) Removal of duplicate or irrelevant facts. Duplicate observations will happen most often during data collection. When you combine data sets from multiple places, scrape data, or

receive data from clients or multiple departments, there are opportunities to create duplicate data. The removal of duplication is one of the largest areas to be considered in this process. Irrelevant observations are when you notice observations that do not fit into the specific problem you are trying to analyze.

B) Fixing structural errors: Structural errors are when you measure or transfer data and notice strange naming conventions, typos, or incorrect capitalization. These inconsistencies can cause mislabeled categories or classes. For example, you may find “N/A” and “Not Applicable” both appear, but they should be analyzed as the same category.

C) Filter unwanted outliers: Often, there will be one-off observations where, at a glance, they do not appear to fit within the data you are analyzing. If you have a legitimate reason to remove an outlier, like improper data entry, doing so will help the performance of the data you are working with. This step is needed to determine the validity of that number. If an outlier proves to be irrelevant for analysis or is a mistake, it may be removed.

Data transformation is the process of converting data from one format or structure into another. Transformation processes can also be referred to as data wrangling, or data munging, transforming, and mapping data from one "raw" data form into the format required at the next step of our methodology – the population of the knowledge graph (triplestore).

3.5 Knowledge graph population (Step #5)

In this step, the pre-processed data (facts) are transformed into semantic triples based on the RDF framework. The transformation and population process rely on the generation of unique, de-referenceable, and HTTP-resolvable URIs for the resulting triples. We use Python and its third-

party RDFLib[32] to transform the extracted facts into RDF data. RDFLib is an RDF library for Python which includes a SPARQL [23] implementation. The library also contains both in-memory and persistent Graph back-ends. RDF data is a graph where the nodes are URI references, Blank Nodes, or Literals. In RDFLib, these node types are represented by the classes URIRef, BNode, and Literal. URIRefs and BNodes can both be thought of as resources, such as a person, a company, a website, etc. A BNode is a node where the exact URI is not known - usually a node with identity only with other nodes. A URIRef is a node where the exact URI is known. In addition to representing some subjects and predicates in RDF graphs, URIRefs are always used to represent properties/predicates and Literals represent object values, such as a name, a date, a number, etc. Using RDFLib we construct the graph in-memory and persistent storage of these graphs are achieved by a triplestore. Being a graph database, triplestores store data as a network of objects with materialized links between them. This makes RDF triplestores the preferred choice for managing highly interconnected data. Triplestores are more flexible and less costly than a relational database and allow for the construction of large knowledge graphs. Such an RDF database often called a semantic graph database, is also capable of handling powerful semantic queries and supports the use of inference services for uncovering new information out of the existing relations.

3.6 Software Analytics Queries (Step #6)

There have been many works on creating software analytics tools or services to support software analysis tasks [57-59]. However, their applicability is often limited for several reasons. Firstly, these tools have remained information silos by relying on their own proprietary data collection and data models. While these models work well for supporting tool-specific analytics tasks, they limit their ability to share, reuse and integrate data and analysis results with other tools and knowledge resources. Secondly, the underlying knowledge representations used by these

[32] https://rdflib.readthedocs.io/

analysis approaches lack support for a seamless integration of new knowledge resources or the ability to deal with incomplete data; and thirdly, the extensibility and flexibility of the analysis support provided by existing tools is often limited by the underlying knowledge representation.

In our research, we take advantage of the Semantic Web and its technology stack (e.g., ontologies, reasoning services) **and establish a unified knowledge graph representation to model software vulnerability-related resources**. Based on this knowledge graph, we can now extend and integrate this knowledge with other (heterogeneous) resources to grant a more **flexible software analytics analysis** approach. Users can either define their software analytics service using SPARQL [23] which is a standard semantic query language to retrieve and manipulate data stored in the RDF format or reuse already predefined queries.

3.7 Knowledge Graph evolution

The last step of our methodology is related to the evolution of the knowledge graph. This is an iterative process within our ontology design that evolves as new vulnerability-related resources become available and will be included in our knowledge model. With the inclusion of new concepts and properties to the domain ontology to capture new vulnerability-related dependencies and semantics, there is a possibility that existing domain concepts and properties will be demoted to the lower layer. One of the key benefits of using ontologies is that they can easily be extended, by adding new relationships and concepts and linking them to the existing ontologies. As a result, our current ontologies will evolve as new knowledge becomes available, however, given the native support of ontologies to support the extension to the knowledge base, these changes will not impact our already existing queries or analysis services. Instead, it will create new opportunities to introduce other novel analysis services and a further extended knowledge base.

Chapter 4

A Knowledge Graph for Software Vulnerability-Related Resources

In this chapter, we illustrate how our methodology (introduced in Chapter 0) can be applied to establish a knowledge graph for the domain of known software vulnerabilities. More specifically, we show how our methodology can be used to create a standardized and unified representation of software vulnerability-related artifacts and knowledge resources, as is shown in Figure 4-1. This unified knowledge representation will then form a base for the introduction of novel software analytics services that can guide developers, maintainers, and security experts in managing known vulnerabilities.

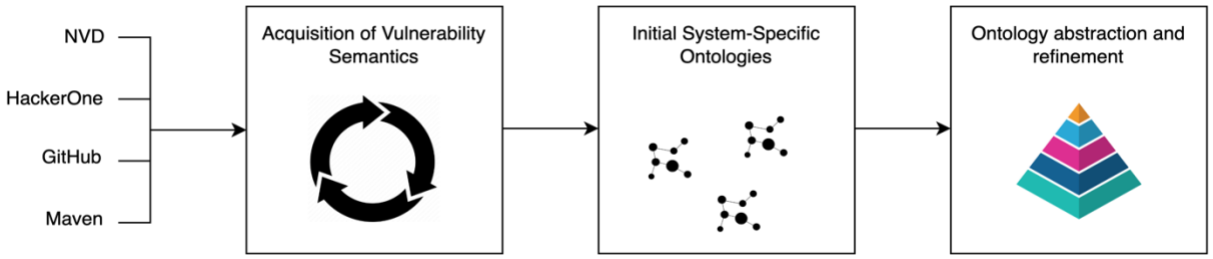

Figure 4-1: Overview picture of the actual process applied to vulnerabilities

4.1 Selection of Software Vulnerability-Related Resources

As previously discussed, various repositories have been introduced to manage known vulnerability information (e.g., NVD, CWE, bug bounty repositories). In addition, other resources such as build and dependency management systems, source code repositories, or even more general information resources (e.g., Wikipedia) can provide useful insights for security experts and developers. While these knowledge resources might not capture directly known software vulnerability information, they can, when integrated with VDBs, provide new insights. To illustrate how our methodology can be applied to support the modeling of these knowledge resources related to known software vulnerability, we select resources that provide different views on software vulnerabilities. We then model these resources and integrate them as parts of our software vulnerability knowledge graph.

Vulnerability Databases

NVD is the most widely used VDB and collects vulnerability information from various interrelated vulnerability databases like CVE, and CWE. Every entry in the NVD database is identified by a unique identifier, a CVE ID, which is also used as a standard reference to NVD vulnerabilities in other resources. A typical vulnerability entry in NVD consists of a vulnerability identifier, a description of the vulnerability, a list of software and their versions in which this vulnerability is found, and a vulnerability severity score (CVSS).

Other than NVD, we included Snyk [5] as a second example of a VDB, which allows us to illustrate how our modeling approach can support and integrate domain knowledge from different resources. The Snyk VDB captures and describes also known vulnerabilities using a CVE ID. In addition, Synk provides services to scan, prioritize, and fix security vulnerabilities in source code. We use the CVE-ID as our main property to link and integrate different VDBs.

Bounty platforms

Bug bounty programs have emerged as an integrated part of the secure software development lifecycle to aid security teams in the release and maintenance phases. Among the many bounty platforms introduced in the last decade, HackerOne was founded in 2013 and is one of the best-known bounty platforms [43]. HackerOne includes two types of information: 1.) the program definition where the organizer publishes bounty rules including policy, scope, reward, etc. 2.) hacker activities (Hacktivity reports) that describe the bounty hunter activities and potential vulnerability discovered by the hacker that needs to be evaluated by the organizer, as well as information about the bounty hunter. Once a new vulnerability is discovered, it might be assigned a CVE-ID.

Dependency Management tools

Dependency and build management tools [60, 61] are used to resolve and manage direct and transitive system dependencies during the build process. Project dependencies can be found within a project source code (e.g., GitHub), a build management system (e.g., Maven), or package managers (e.g., NPM). To populate our ontology, we use another service called libraries.io[33] that monitors packages across 32 package managers while it provides an API to search and extract dependencies of libraries monitored by the platform.

3rd Party Knowledge graphs or Linked data

One of the key benefits of knowledge graphs is the ability to integrate already existing knowledge graphs or linked data resources, to allow users (e.g., humans or machines) to explore and infer new knowledge across a set of combined knowledge resources. We illustrate this integration of already existing linked data and knowledge graphs by linking our own knowledge graph with DBpedia, which is the linked data representation of the Wikipedia portal. These third-party knowledge

[33] https://libraries.io/

integrations facilitate establishing a direct link between the knowledge resources using shared concepts or properties. DBpedia is a publicly available linked data resource that provides a SPARQL query endpoint to access the knowledge graph, We also attempted to integrate with the Google Knowledge Graph [33]. However, the Google Knowledge Graph uses a RESTful search API, limiting the ability to fully integrate this knowledge graph semantically with other knowledge graphs.

4.2 Establishing Initial System-Level Ontologies

In this section, we discuss the initial system-level ontologies, which we created as part of our modeling approach. We established these initial system-level ontologies, by transforming the extracted schemata into their corresponding ontology components, namely concepts, properties, and relations. It should be noted that these initial system-level ontologies are neither optimized in terms of their semantic expressiveness (reasoning support) nor for knowledge reuse and linking across ontologies.

4.2.1 Vulnerability-Related Ontologies

The vulnerability resources for which we derived system-level ontologies are CWE, CPE, and two VDBs (NVD and Snyk) which are explained as follows.

CWE

The CWE is a category system for hardware and software weaknesses and vulnerabilities which is available on the Mitre website[34]. The downloaded file is an XML that holds all the metadata available for weaknesses. At the time of writing, there are 947 weaknesses listed in CWE V4.6.

[34] https://cwe.mitre.org/data/downloads.html

Figure 4-2: shows a sample CWE record with all the available properties used to describe weaknesses.

Each CWE has several primitive attributes (e.g., name, date, status, etc.) that we store as a data attribute as well as some relational attributes that link weaknesses to each other. For instance, CWE-943 (Improper Neutralization of Special Elements in Data Query Logic) is a parent of CWE-89 (SQL Injection), and CWE-89 is a parent of CWE-564 (SQL Injection: Hibernate). A query for vulnerability with a weakness CWE-943 will now not only return vulnerabilities with CWE-943 but also vulnerabilities in all its subcategories CWE-89 and CWE-564.

```
<Weakness ID="1041" Name="Use of Redundant Code" Abstraction="Base" Structure="Simple" Status="Incomplete">
        <Description>The software has multiple functions, methods, procedures, macros, etc. that
          contain the same code.</Description>
        <Extended_Description>
            <xhtml:p>
                This issue makes it more difficult to maintain the software, which indirectly affects security
                by making it more difficult or time-consuming to find and/or fix vulnerabilities.
                For example, if there are two copies of the same code, the programmer might fix a weakness
                in one copy while forgetting to fix the same weakness in another copy.
            </xhtml:p>
        </Extended_Description>
        <Related_Weaknesses>
           <Related_Weakness Nature="ChildOf" CWE_ID="710" View_ID="1000" Ordinal="Primary"/>
        </Related_Weaknesses>
        <Weakness_Ordinalities>
           <Weakness_Ordinality>
              <Ordinality>Indirect</Ordinality>
           </Weakness_Ordinality>
        </Weakness_Ordinalities>
        <Common_Consequences>
           <Consequence>
              <Scope>Other</Scope>
              <Impact>Reduce Maintainability</Impact>
           </Consequence>
        </Common_Consequences>
        <Taxonomy_Mappings>
           <Taxonomy_Mapping Taxonomy_Name="OMG ASCMM">
              <Entry_ID>ASCMM-MNT-19</Entry_ID>
           </Taxonomy_Mapping>
        </Taxonomy_Mappings>
        <References>
           <Reference External_Reference_ID="REF-960" Section="ASCMM-MNT-19"/>
        </References>
        <Content_History>
           <Submission>
              <Submission_Name>CWE Content Team</Submission_Name>
              <Submission_Organization>MITRE</Submission_Organization>
              <Submission_Date>2018-07-02</Submission_Date>
              <Submission_Comment>Entry derived from Common Quality Enumeration (CQE) Draft 0.9.</Submission_Comment>
           </Submission>
        <Modification>
          <Modification_Name>CWE Content Team</Modification_Name>
          <Modification_Organization>MITRE</Modification_Organization>
          <Modification_Date>2020-08-20</Modification_Date>
          <Modification_Comment>updated Relationships</Modification_Comment>
        </Modification>
        </Content_History>
      </Weakness>
```

Figure 4-2: CWE XML Schema (partial view).

Our CWE system-level ontology (shown in Figure 4-3:) is a direct mapping from the CWE XML schema. In addition, we introduce two concepts which are subclasses of the General concept of Software Weaknesses. The concept of *CWE Attributes* and its subclasses capture general information related to the weakness (e.g., operating system, technology, languages). These CWE Attributes are mostly derived from main concepts in other ontologies with their logics. We reuse these concepts to primarily connect our ontology to the linked data and secondly, to use predefined data in our ontology. The *CWE* concept uses *CWE Attributes* as object properties and holds other attributes (e.g., *submission date*) as data properties.

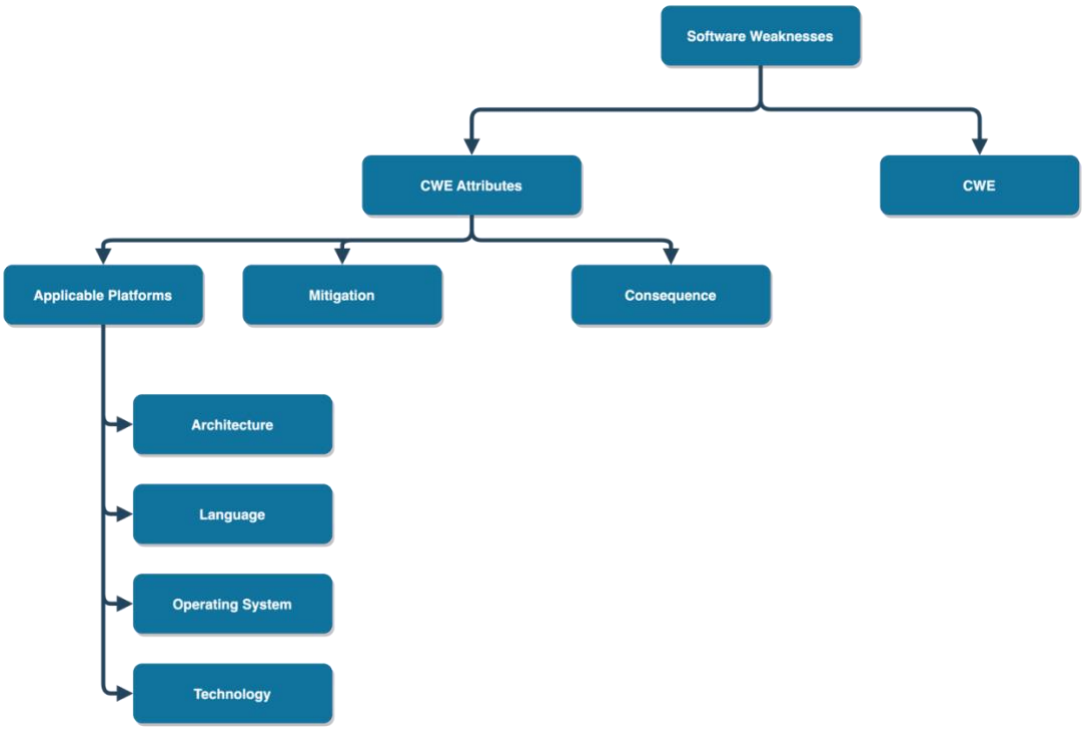

Figure 4-3: CWE - System Level Ontology

Figure 4-4 shows a partial view of our populated CWE ontology and how it captures the dependencies among CWEs. It should be noted that the relations (e.g., parentOf/childOf) are inverse. Properties have a direction from domain to range. The owl:inverseOf construct can be used to define a symmetric property.

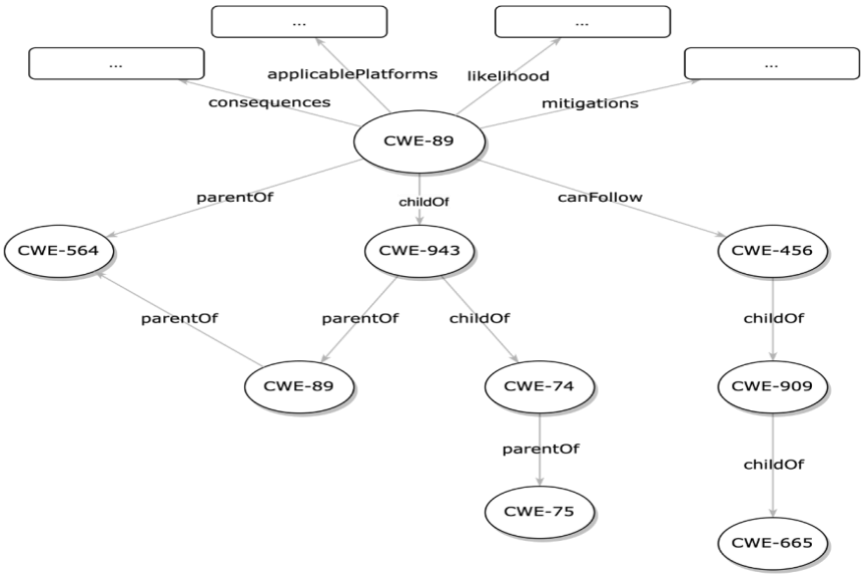

Figure 4-4: A partial ABox view of the initial CWE system ontology

CPE

CPE represents a directory of projects and their versioning information. The directory was introduced to eliminate ambiguity among project references.


```
{
        "deprecated": false,
        "cpe23Uri": "cpe:2.3:a:uncurl_project:uncurl:0.03:*:*:*:*:*:*:*",
        "lastModifiedDate": "2021-04-13T13:39Z",
        "titles": [
            {
                "title": "Uncurl Project Uncurl 0.03",
                "lang": "en_US"
            }
        ],
        "refs": [
            {
                "ref": "https://github.com/chrisd1100/uncurl/releases",
                "type": "Version"
            },
            {
                "ref": "https://github.com/chrisd1100/uncurl",
                "type": "Project"
            }
        ],
        "deprecatedBy": [],
        "vulnerabilities": []
    }
```


Figure 4-5: A sample CPE record

Each CPE has a unique, standardized URI that can be de-referenced for later processing. The URI combines 13 attributes, separated by a colon and consists of the following parts.

<standard>:<standardVersion>:<part>:<vendoe>:<product>:<version>:
<update>:<edition>:<language>:<software_edition>:<target_software>:
<target_hardware>:<other>

For our CPE system-level ontology, we model these attributes as properties and attributes of our CPE class.

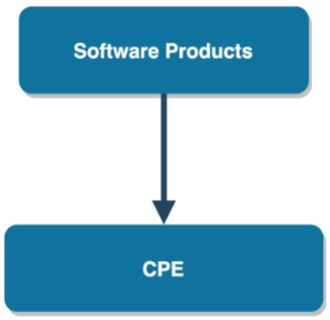

Figure 4-6: Initial CPE System Level Ontology

As shown in Figure 4-5: each CPE includes a list of references in addition to URI and titles. These references are URIs to the product-related webpages, including in many cases the link to the source code repository.

Figure 4-6: shows the corresponding CPE system level ontology, where the `Software Products` concept captures the CPE title and the core CPE properties, as well as the detailed reference to each CPE. It should be noted that in Figure 4-6: we omitted properties and attributes. In the original CPE repository, many products have more than one URI due to having different versions or environments in a component that might be deployed. To facilitate further processing, we transform CPE information and extract a reference that is common for all different versions of a single product. The software product will provide a standardized reference to the software

product, while the CPE instance, will include the CPE specific version information (Figure 4-6:). Figure 4-7: shows a partial ABox view of the initial CPE system ontology.

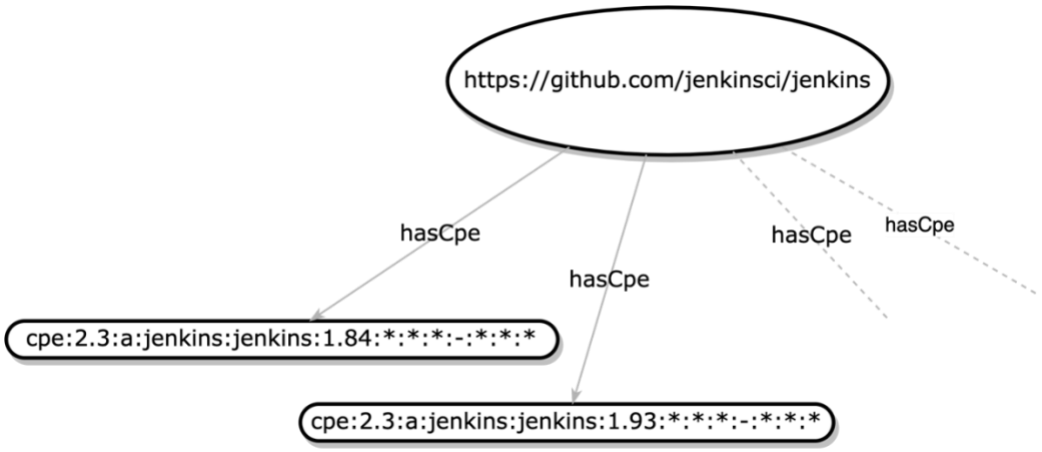

Figure 4-7: A partial ABox view of the initial CPE system ontology

NVD

NVD is the most widely used publicly available VDB for known vulnerabilities. Figure 4-8 shows a sample of NVD record. NVD uses CVE-ID to identify vulnerabilities (CVEs) and maintain a reference to the original record by MITRE. MITRE Engenuity (or simply Engenuity) was launched in 2019 "to collaborate with private sectors on solving industrywide problems with cyber defense" in collaboration with corporate partners [62]. The foundation created the Center for Threat-Informed Defense. In addition, each CVE includes a problem-type section that lists all weaknesses from the CWE database that apply to this CVE. The reference property includes a list of external web resources that contain additional information about the vulnerabilities. The configurations property is used to match a vulnerability to a list of CPE records and a software product (including the affected versions, platforms, and other product-specific attributes).

```
{
    "cve": {
        "data_type": "CVE",
        "data_format": "MITRE",
        "data_version": "4.0",
        "CVE_data_meta": {
            "ID": "CVE-2021-33477",
            "ASSIGNER": "cve@mitre.org"
        },
        "problemtype": {
            "problemtype_data": [
                {
                    "description": [
                        {
                            "lang": "en",
                            "value": "CWE-755"
                        }
                    ]
                }
            ]
        },
        "references": {
            "reference_data": [
                {
                    "url": "https://git.enlightenment.org/apps/eterm.git/log/",
                    "name": "https://git.enlightenment.org/apps/eterm.git/log/",
                    "refsource": "MISC",
                    "tags": [
                        "Third Party Advisory"
                    ]
                }
                    ...
            ]
        },
        "description": {
            "description_data": [
                {
                    "lang": "en",
                    "value": "rxvt-unicode 9.22, rxvt 2.7.10, mrxvt 0.5.4, and Eterm 0.9.7 allow ..."
                }
            ]
        }
    },
    "configurations": {
        "CVE_data_version": "4.0",
        "nodes": [
            {
                "operator": "OR",
                "children": [],
                "cpe_match": [
                    {
                        "vulnerable": true,
                        "cpe23Uri": "cpe:2.3:a:eterm_project:eterm:0.9.7:*:*:*:*:*:*:*",
                        "cpe_name": []
                    }
                            ...
                ]
            }
        ]
    },
    "impact": {
        "baseMetricV3": {
            "cvssV3": {
                "version": "3.1",
                "vectorString": "CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H",
                  ...
                "baseSeverity": "HIGH"
            },
            "exploitabilityScore": 2.8,
            "impactScore": 5.9
        }
              ...
    },
    "publishedDate": "2021-05-20T20:15Z",
    "lastModifiedDate": "2021-05-30T19:15Z"
}
```


Figure 4-8: Partial view of the CVE Schema

For our initial system-level NVD ontology (shown in Figure 4-9), we map a subset of the available NVD information that is required for our use cases. The modelled information includes the following data properties: CVE-ID, publish date, and score, as well as object properties such as CWEs, CPEs. The current ontology can easily be extended to include all NVD information.

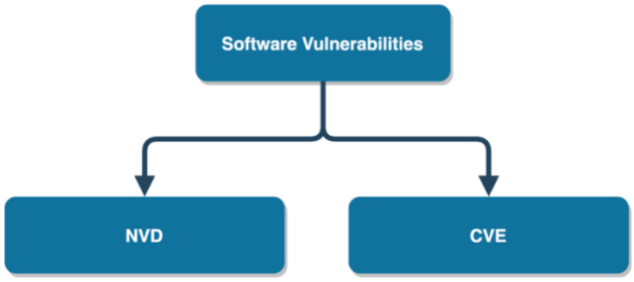

Figure 4-9: NVD System Level Ontology

Figure 4-10 shows a partial ABox view of our populated NVD system-level ontology and the information we capture in this ontology.

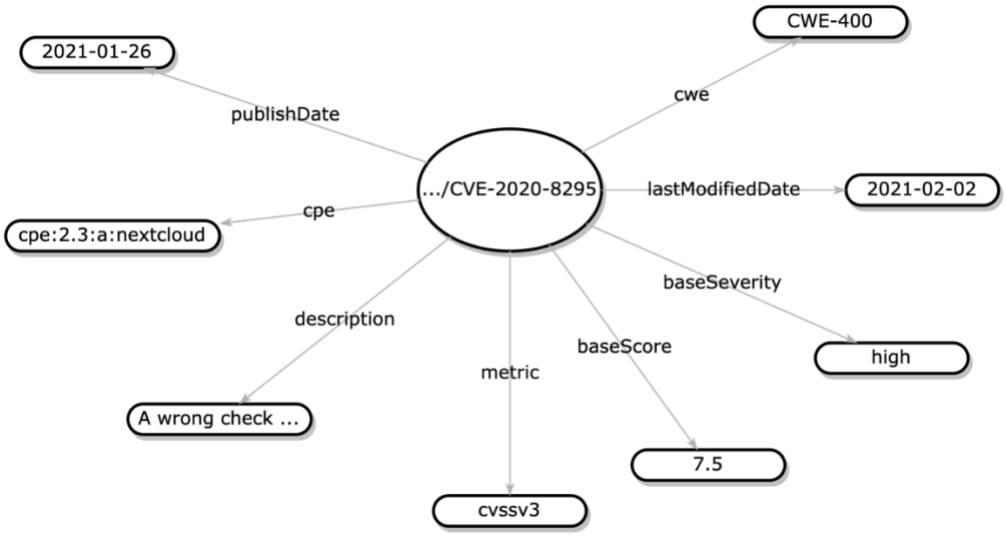

Figure 4-10: A partial ABox view of the initial CVE system ontology

Snyk

Snyk data schema is similar to the one use by NVD by including both CVE and CWE. Snyk provides its own scoring system to classify and measure risk factors. Snyk does not offer an API endpoint to fetch vulnerabilities. The extraction of vulnerability information in Synk is limited to the portal website, by scraping the web pages. Figure 4-11 illustrates how the Snyk system level ontology relates to other ontologies in our ontology hierarchy.

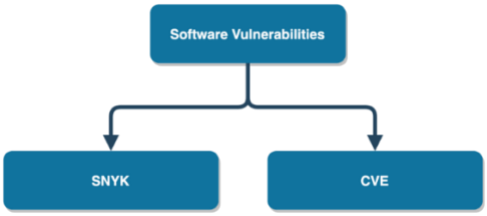

Figure 4-11: Snyk initial design

Figure 4-12 shows a partial ABox view of our populated *SNYK* system-level ontology and the information it captures.

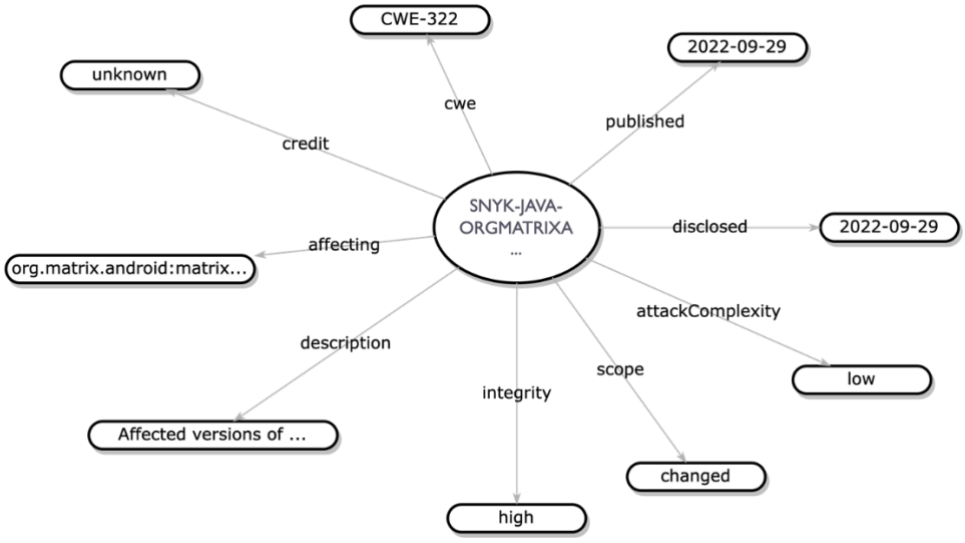

Figure 4-12: Snyk initial example

HackerOne

Bug bounty programs offer monetary rewards to ethical hackers (bounty hunters) for successfully discovering and reporting a vulnerability or bug to an application's developer. Figure 4-13: shows a partial view of a sample record (using "Nextcloud") that is available on the HackerOne portal. Each record on HackerOne contains its status and some minimum reward information, as well as general bounty descriptions that include the program being part of the bounty. The hacker activity for a particular bounty is posted as HackerOne hacktivity reports, which contain the title, date, status, and reporter. In the HackerOne model, we have one program entity and one hacktivity entity to map to projects. We also model users who reported a potential vulnerability as bounty hunters in this ontology.


```
{
    "id": 13291,
    "url": "/nextcloud",
    "name": "Nextcloud",
    "meta": {
        "submission_state": "open",
        "resolved_report_count": 292,
        "minimum_bounty": 100,
        "default_currency": "usd",
        "offers_bounties": true,
        "quick_to_bounty": false,
        "quick_to_first_response": true
    },
    "about": "Access, share and protect your files, calendars, ...",
    "stripped_policy": "As an open-source project we know and ...",
    "handle": "nextcloud",
    "profile_picture": "https://profile-photos.hackerone-user-...",
    "internet_bug_bounty": false,
    "team_type": "team"
}
```


Figure 4-13: A sample HackerOne record

HackerOne does not provide an official API for public access to its programs and to create data dumps. The website[35] therefore needs to be crawled to extract the reported vulnerability and hacktivity information. The challenge with scraping the website is that the content provided for each hacktivity report is not standardized, making it difficult to extract consistent data from each report.

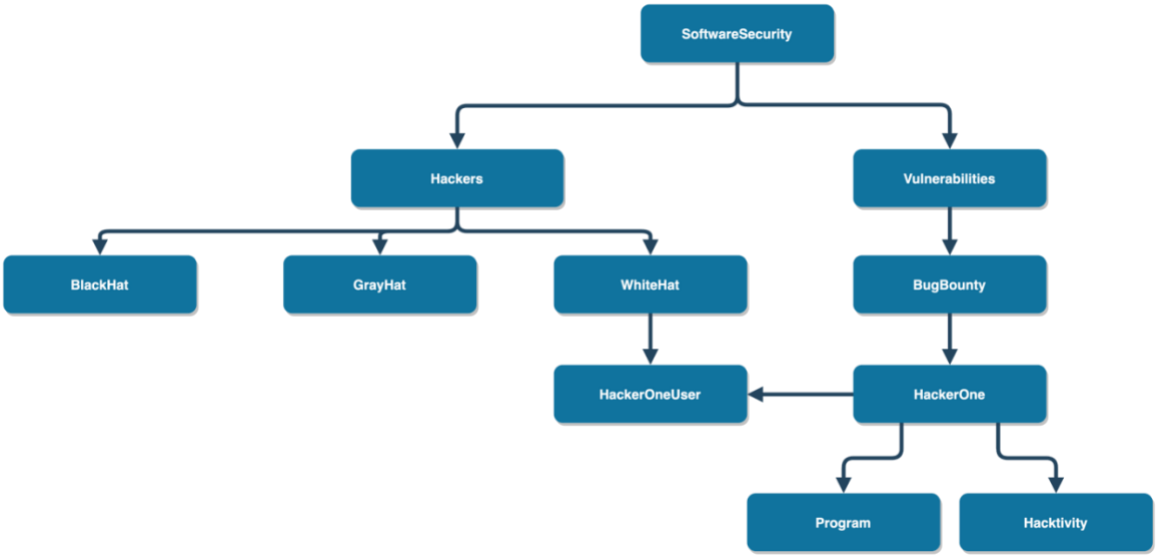

Figure 4-14: Bug bounty System-level ontology

The initial design of our system-level ontology for HackerOne is shown in Figure 4-14. We face a few challenges when creating the initial `HackerOne` concept. While extracting some information related to the bounty concept (policy, collection of hackers reports) was a

[35] https://hackerone.com/

straightforward task, extracting the content of the Hacktivity reports is quite more difficult due to their unstructured representation (e.g., specifying details of the vulnerable components/systems such as versioning information). Also, there is no consistency in terms of the type of information provided between Hacktivity reports. Therefore, we treat most of the descriptions found in a Hacktivity report currently as a single text property.

4.2.2 Build Dependency Ontology

Libraries.io

Libraries.io[36] is a service portal that indexes data from 5 million packages extracted from 32 package managers. The website monitors package releases, analyses each project's code, community, distribution, and documentation and also maps relationships between packages when they are declared as a dependency.

This type of dependency information is usually accessible through build management tools (e.g., Maven), package managers (e.g., NPM), or source code repositories (e.g., GitHub). For the build management ontology design, we reuse the build ontology introduced originally in [63], which provides support for the modeling and integration of different build management systems.

For the data extraction process, instead of querying each build management system separately, we used the libraries.io portal. Libraries.io is an open-source web service that lists software development project dependencies. Figure 4-15 shows a partial record of the information libraries.io provides for each individual product. The extracted information was then converted to our graph schema for import (Figure 4-16). In addition, libraries.io also provides another API to retrieve product dependencies.

[36] https://libraries.io

```
{
    "dependent_repos_count": 515312,
    "dependents_count": 151825,
    "deprecation_reason": null,
    "description": "React is a JavaScript library for building user interfaces.",
    "forks": 37364,
    "homepage": "https://reactjs.org/",
    "keywords": [
        "react",
        "declarative",
        "frontend",
        "javascript",
        "library",
        "ui"
    ],
    "language": "JavaScript",
    "latest_download_url": "https://registry.npmjs.org/react/-/react-18.0.0-rc.1-next-f468816ef-20220225.tgz",
    "latest_release_number": "18.0.0-rc.1-next-f468816ef-20220225",
    "latest_release_published_at": "2022-02-28T16:13:09.862Z",
    "latest_stable_release_number": "17.0.2",
    "latest_stable_release_published_at": "2021-03-22T21:56:19.536Z",
    "license_normalized": false,
    "licenses": "MIT",
    "name": "react",
    "normalized_licenses": [
        "MIT"
    ],
    "package_manager_url": "https://www.npmjs.com/package/react",
    "platform": "NPM",
    "rank": 34,
    "repository_license": "MIT",
    "repository_url": "https://github.com/facebook/react",
    "stars": 182992,
    "status": null,
    "versions": [
        {
            "number": "0.0.0-00d4f95c2",
            "published_at": "2021-03-15T16:12:35.231Z",
            "spdx_expression": "MIT",
            "original_license": "MIT",
            "researched_at": null,
            "repository_sources": [
                "NPM"
            ]
        },
        {
            "number": "0.0.0-0203b6567",
            "published_at": "2021-03-16T16:13:04.042Z",
            "spdx_expression": "MIT",
            "original_license": "MIT",
            "researched_at": null,
            "repository_sources": [
                "NPM"
            ]
        },
        {
            "number": "0.0.0-0935a1db3",
            "published_at": "2021-02-03T18:09:51.735Z",
            "spdx_expression": "MIT",
            "original_license": "MIT",
            "researched_at": null,
            "repository_sources": [
                "NPM"
            ]
        }
    ]
}
```


Figure 4-15: Data extracted from library.io

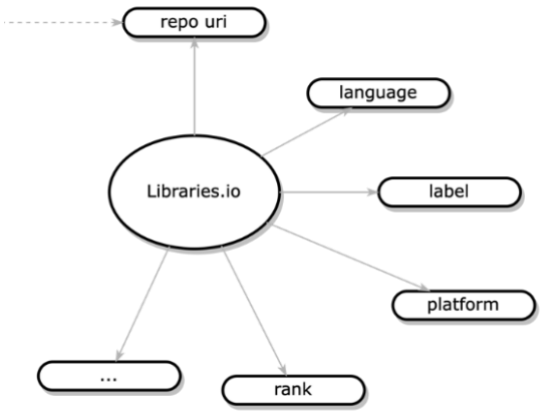

Figure 4-16: A partial ABox view of the initial Libraries.io system ontology

Figure 4-17 shows an example of the dependencies that we extracted and modeled for the sample library "deep-defaults". For each project, we retrieve these transitive dependencies, with the outgoing connections (edges) being dependencies and incoming connections (edges) being dependents. It should be noted that for the dataset which we used in our case studies, we limited the transitivity dependencies to 5 nesting levels.

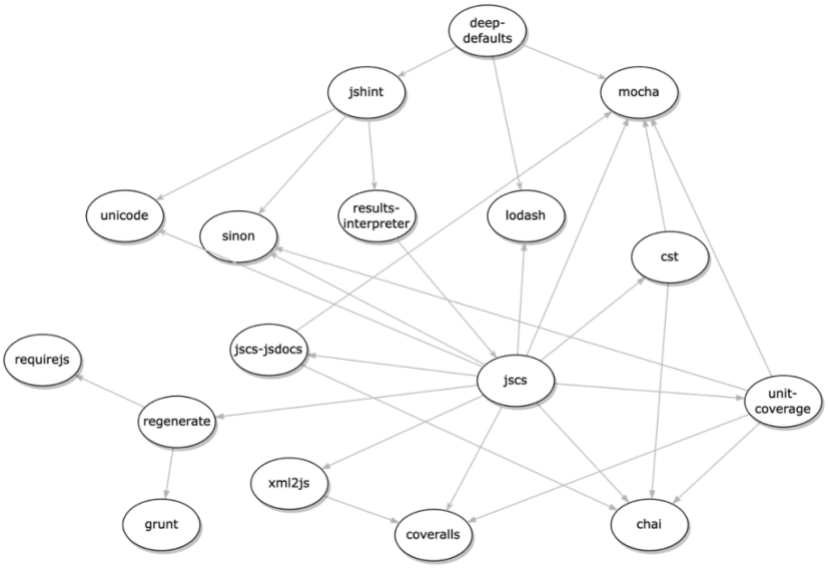

Figure 4-17: Dependencies for "deep-defaults" library

4.3 Ontology Abstraction and Refinement

As discussed earlier, this thesis contributes by not only modeling individual knowledge resources as ontologies, but also by integrating these ontologies in a unified representation (knowledge graph). As a part of our modeling approach, we take the advantage of key premises of SW which are its ability to share and extend the existing knowledge. The resulting unified representation allows us to eliminate the information silos these resources have traditionally remained in and turn them into information hubs. These information hubs enable applications and analytics services to reuse and share their knowledge across individual repository boundaries.

In this section, we discuss in more detail, how we integrate our initial system-level ontologies (Section 4.2). More specifically, we refine our system-level ontology designs to facilitate their integration with the ontologies introduced by Alqahtani et al. [7]. They introduced SEVONT, an abstraction hierarchy of ontologies covering various software artifacts, including ontologies for software vulnerability databases [7]. SEVONT also proposes a unification model for VDBs using linked data that facilitates also the reconciliation of VDBs. In what follows, we discuss in more details, the modifications and optimization we made to our initial system ontology designs, as well as changes to the SEVONT framework in order to integrate our ontologies. More specifically, we not only extend these ontologies with additional system-level repositories (e.g., bounty resources) and non-specialized resources (e.g., DBpedia), but also refine the ontology design to provide a knowledge graph that goes beyond the conceptualization of a domain of discourse by focusing on the inference of new knowledge to support our use cases.

4.3.1 Abstraction Overview

For the modeling of abstractions, we followed the same bottom-up knowledge modeling approach used in [43]. First, we modeled system-specific concepts (Section 4.2) which we now evaluate and refine to determine if certain concepts can be promoted to higher-level shared

concepts that finally will be located in upper-level ontologies (Figure 4-18). This abstraction mechanism allows not only for the reuse of concepts and properties across ontologies, but also facilitates linking (shared concepts) of the ontologies. The resulting four-layer modeling hierarchy is similar to a metadata modeling approach introduced by the Object Management Group (OMG)[37].

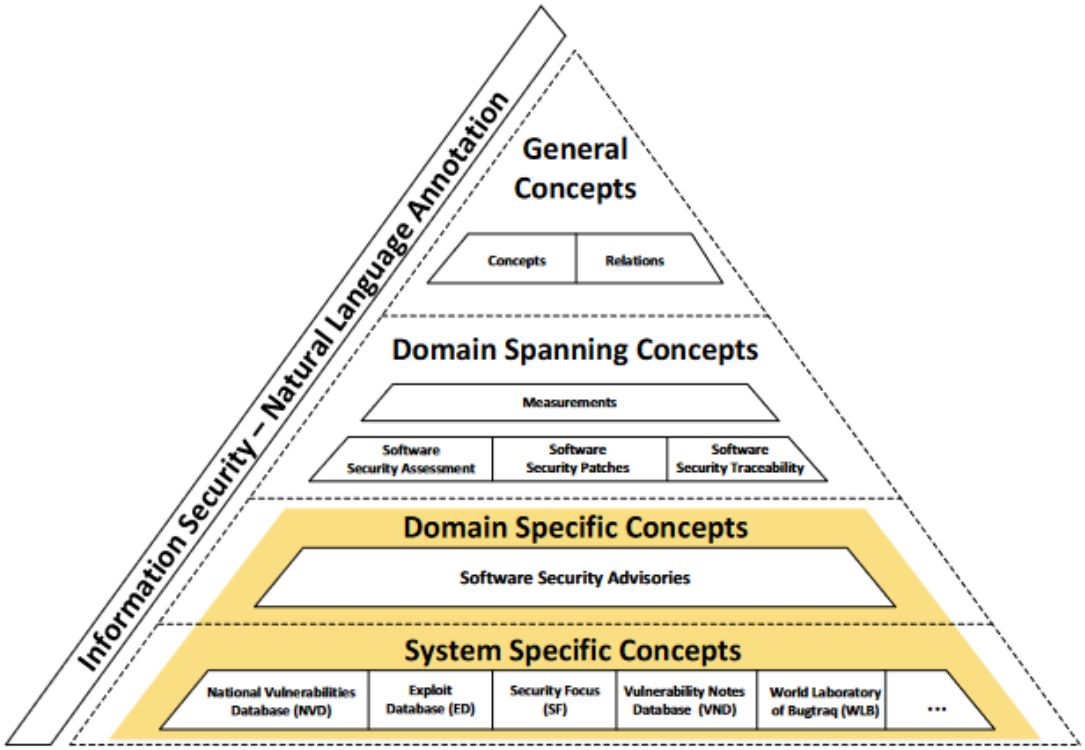

Figure 4-18: SEVONT Ontology Pyramid [65]

[37] http://www.omg.org/

4.3.2 Design Details

In what follows, we discuss how each of these layers differs in terms of their purpose and design rationale. The discussion is based on the original ontology abstraction hierarchy presented in [64].

General Concept Layer

Classes in the top-layer model correspond to meta-meta level ontologies—core concepts shared and extended by the lower modeling layers. Examples of such core concepts are `Product`, `Organization`, `Activity`, `Stakeholder`, and `Artifact`. All concepts in this layer are subclasses of the `SeOn Thing` class (a subclass of owl:Thing, which captures the set of all individuals within our framework). Similarly, the datatype properties and object properties in this layer are generic and shared across the abstraction layers. For example, the `dependsOn` object's property captures the generic relationship between things—one Product depends on another Artifact.

Domain-Spanning Concepts

In this layer, concepts describe the knowledge that is typically inferred from two or more ontologies. For example, the measurements ontology acts as a general linking mechanism between ontologies. The ontology includes two basic concepts, BaseMeasure and DerivedMeasure. Adequate BaseMeasure instances are the size and numberOfDependencies in a Product. DerivedMeasure captures an aggregation of values from different subdomains. For example, the DerivedMeasure class includes the numberOfVulnerabilitiesPerLibrary instance, which is computed from measures collected from the source code, history, build system, and vulnerability ontologies. SimilarityMeasure, which is a subclass of DerivedMeasure, captures the similarity ([0,1]) between any two SeonThing instances.

Domain-Specific Concepts

This layer in the knowledge model captures domain-specific aspects and concepts that are common and reused across system-level resources in a particular domain (e.g., domain of dependency management systems). At the core of the domain-specific layer, we have several domain ontologies: (1) SEVONT, (2) Software Evolution Ontologies (SEON) [64], (3) Software Build Systems Ontologies (SBSON), and (4) Bug bounty ontologies.

Our ontology includes the following important domain concepts:

- Vulnerability. In software security, a vulnerability refers to a flaw in the system that is introduced by reusing vulnerable (external) software components or inadvertent coding mistakes by developers (e.g., bad coding practices).
- Product. Software products are assets of organizations that are the results of a software development process (e.g., hardware, artifacts).
- Attacker. Attackers, either internal or external entities of the system, attack a product to perform malicious actions which attempt to break the security of a software system or its components.
- Attack. Attacks are malicious actions designed to compromise the security of a system. Security experts analyze these attacks to study the behavior of attackers, estimate the cost of attacks and determine their impact on overall system security.
- Countermeasure. A countermeasure is a mechanism used to protect a system from potential vulnerability attacks (e.g., patch development, encryption/decryption enhancement, and updated system security configurations).

For example, security databases capture a Vulnerability that has an associated Event. An Event often can be further divided into Action and Impact—an attacker exploits a Vulnerability to produce an Action, which has an Impact.

System-Specific Concepts

The bottom layer defines concepts that are specific to a knowledge resource and not shared among other ontologies. The system ontologies extend domain-specific concepts. For example, the system ontology for *NVD* extends the general SEVONT ontology with a *Severity* concept, which is specific to NVD and is not shared among other VDBs.

4.3.3 Extensions to the SEVONT Hierarchy

In what follows, we discuss the changes we made to our initial system-level ontologies to support not only their integration in the existing SEVONT ontologies, but also improve knowledge reuse and inference of new knowledge. As a part of our modeling process, we extend the layers of the original ontology hierarchy by identifying and promoting newly shared concepts (introduced by our system-level ontologies) to a higher abstraction layer. Our proposed model reduces concept ambiguity and facilitates ontology alignment/matching by determining correspondences between ontological concepts. The resulting linked ontologies form a foundation to support the introduction of novel analytics services that can guide maintainers and developers in managing known software vulnerabilities. An overview of our modified four-layered SEVONG ontology abstraction model is shown in Figure 4-19 and is discussed as follows.

General Abstraction Layer

For the general abstraction layer, there are no changes to the original ontology design from [53] since the provided general abstraction layer already covers the most basic core concepts. Our system-level ontologies are all based on the extensions of these already modeled general concepts.

Domain Spanning Abstraction Layer

We extended the domain Spanning layer with an activity ontology consists of two concepts: *activity* and *reward*. These two domain-spanning concepts are not limited to any domain and can be reused across many areas. All bounty reports can be considered as results of some

activities made by stakeholders with a potential reward. In our context, they are used to capture reports and efforts in the bug-bounty domain.

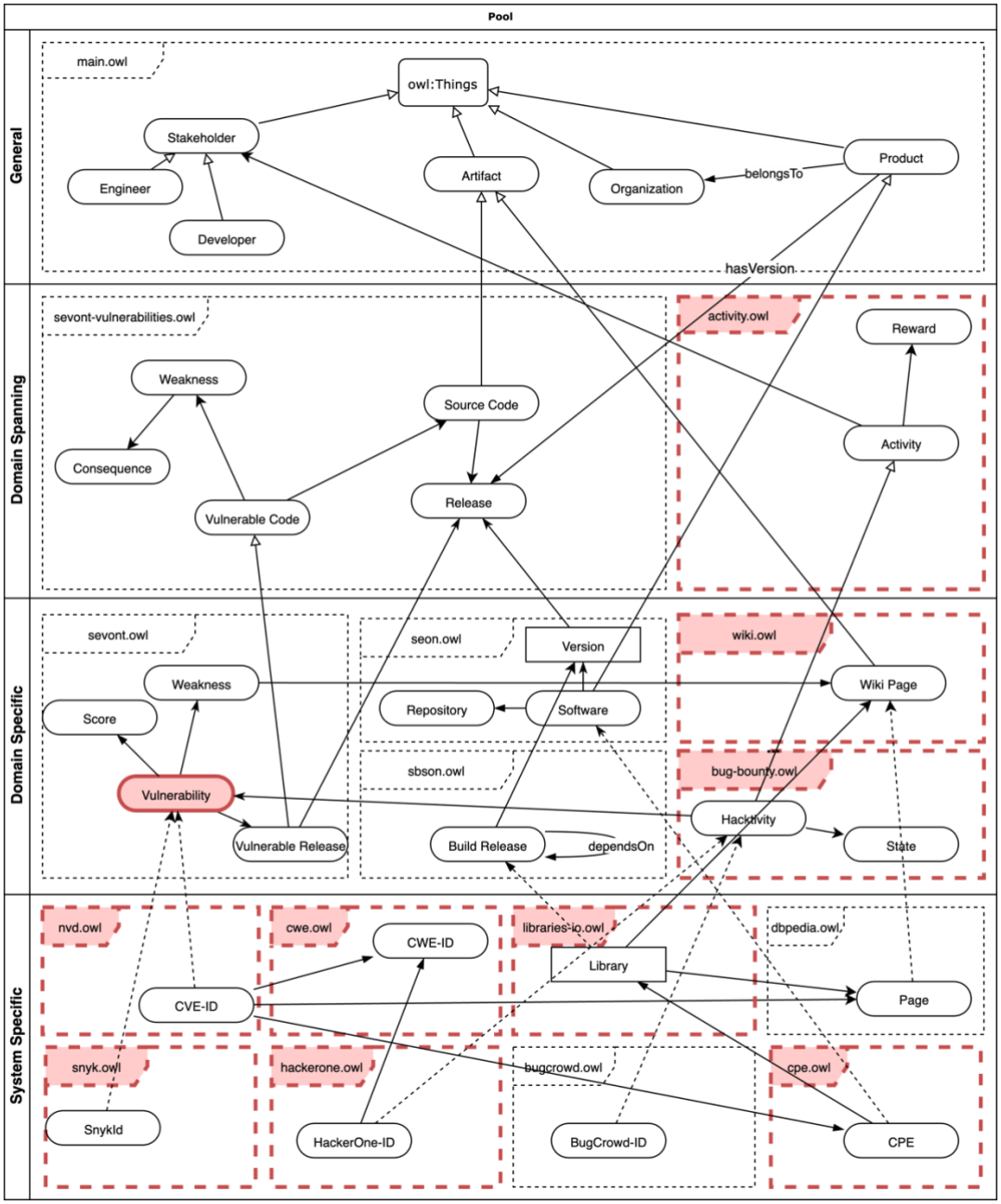

Figure 4-19: Four-layer ontology abstraction model

Domain-Specific Abstraction Layer

At the domain-specific layer, we added two domain-level ontologies:

(1) Bug Bounty ontology, and

(2) a wiki ontology.

These two domain ontologies, model the core concepts found in the Wiki and bug-bounty domain. The wiki ontology is an abstract layer for any encyclopedia where a user might find relative information about a subject. It includes the concept *`WikiPage`* and some core properties and attributes. The bug-bounty ontology models the concepts shared among ontologies in the bug-bounty domain (e.g., *`HackerOne`* or *`BugCrowd`*.). Bounty reports are quite common in bounty domain and contain detailed descriptions of vulnerabilities that are disclosed to the program handlers only and allow them to fix the issues before get published publicly. The report usually contains steps to reproduce proof of concept and recommended solutions that leads to identify a vulnerability. The domain ontology, therefore, includes two major concepts:

- `Hacktivity` is used to report on hacker activity. It also serves as a resource that enables hackers to search for reports regarding programs and weaknesses and how these weaknesses were exploited in various bounty programs.
- *`Status`* is a concept related to *`Hacktivity`* and is used to keep track of the status of individual activities.

System-Specific Layer

At the System-Specific layer, the most concrete layer of our hierarchy, knowledge specific to a particular knowledge resource is modeled. For example, SBSON [65] models the common concepts found in the domain of build management systems. There are build system-specific ontologies, like Maven, which extend the domain level SBSON to model information specific to Maven.

Build Management Dependency Graph

Several build systems exist which provide the users access to both internal components and external API dependencies. While their traditional unidirectional dependency models capture build dependencies, they restrict users ability to further reason upon this knowledge. For example, using the case of Maven, it is currently not possible for a user to identify all components or projects that depend either directly or indirectly on a project (Figure 4-20).

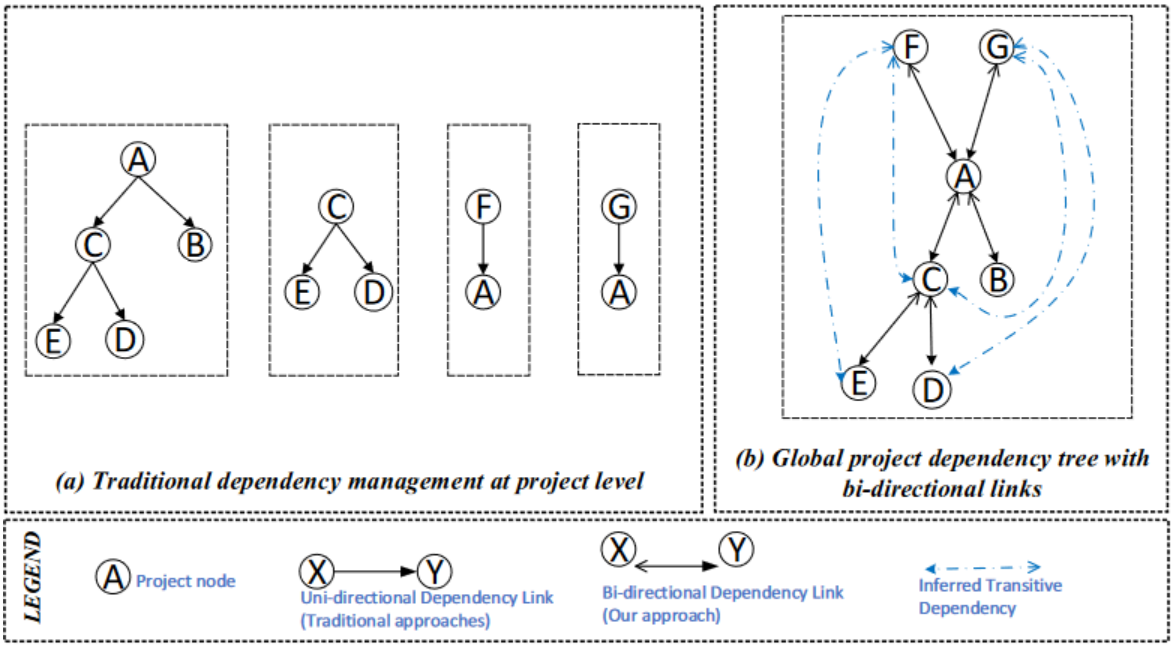

Figure 4-20: Unidirectional vs. bi-directional dependencies. [63]

To overcome the previously discussed modeling challenges, we take the advantage of SBSON ontology [63] to model the dependencies in the build systems and source code repositories. SBSON extends the initial system-level ontology with support for a bi-directional project dependency graph. This bi-directional graph allows us to not only query for all components that a project is using (directly or indirectly), but also to identify which projects are directly or indirectly using a certain component. For example, using the SBSON dependency model (Figure 4-20). We can extend now traditional dependency-based impact analysis on project C to not

include all components on which project C depends on (projects D and E), but also identify all projects which might depend on project C - in this example, projects A, F, and G.

Given this ontology design, it is possible to have system-level ontologies that benefit from these bidirectional links by being able to infer knowledge across knowledge resource boundaries. For example, the system-level Library ontology (shown in Figure 4-21) extends the SBSON ontology, which now allows the Library ontology to also take advantage of the bi-directional dependency model captured in SBSON.

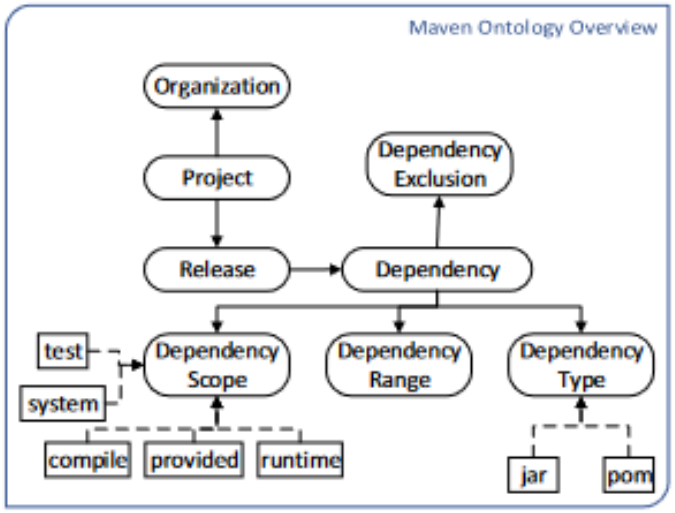

Figure 4-21: Maven Ontology Overview

In what follows. we discuss in more details, some of the design modifications we applied to both our initial system-level ontologies (Section 4.2) and SEVONT.

Linking of Libararies.io with Software Product information (CPE.on)

A fundamental part of the vulnerability analysis process is to be able to uniquely identify products affected by a given vulnerability. As a result of this analysis, we can now identify known issues for any software product, as long as the associated product identifier is known. In our modeling approach, both ontologies, libraries.io and CPE, have a concept `repo_uri` which refers to a software product. We use this concept to link the libraries.io with the CPE ontology.

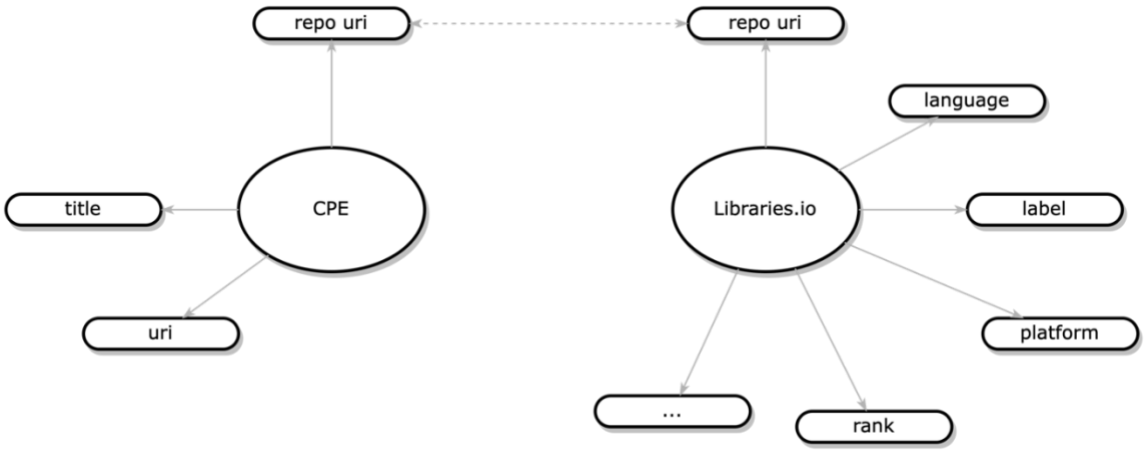

Figure 4-22. CPE-Library integration model

Linking of Weaknesses (CWE) with build dependencies (SBSON)

Based upon the requirements discussed in section 4.2.1 we use the same technique as the Dependency Graph section to utilize the reasoning service and infer all possible weaknesses. In the CWE ontology, we define properties (provided in Table 4-1) with OWL features. The inclusion of these property types allows us to provide a knowledge model which is expressive enough to allow for the inference and linking of knowledge across these two ontologies.

Type of property	Type of reasoning supported
canAlsoBe	Transitive Symmetric
childOf	Transitive Inverse Of "parentOf"
parentOf	Transitive Inverse Of "childOf"
peerOf	Transitive Symmetric

Table 4-1: CWE reasoning details

Linking Bug Bounty information with known vulnerabilities databases

We enrich our system-level VDB knowledge graphs by linking them with CWE and bug bounty ontologies. These links allow us to infer new knowledge which might not have been directly accessible in the individual knowledge resources. The bug bounty repositories include vulnerability-related information in the form of hacktivity reports covering potential vulnerabilities. These vulnerabilities are disclosed by bounty hunters to the organization or vendor who issued the bounty program. However, several inconsistencies can be observed in these reports:

1) When a hacktivity report is verified as describing a vulnerability, this vulnerability might be either disclosed in a vulnerability database (which assigns a CVE-ID to the vulnerability) or just reported locally on the bug bounty platform (without a CVE-ID).
2) Even if a reported vulnerability has been assigned a CVE-ID in the VDB, the corresponding hacktivity report might not be updated with the new CVE-ID.

Therefore, a vulnerability might be reported in a bounty program with or without a CVE-ID, even if the vulnerability has been published in the VDB with a CVE-ID. As a result, using the shared concept `CVE-ID`, while offering high link precision, the recall which can be achieved might be quite low.

```
{
    "cveId": "CVE-2020-8295",
    "hackerOneUrl": "https://hackerone.com/reports/812754",
    "id": 583,
    "refId": "812754",
    "hackerOneHandle": "nextcloud",
    "hackerOneSeverityRating": "high",
    "hackerOneSeverityScore": 7.5,
    "hackerOneSubstate": "resolved",
    "hackerOneBountyCurrency": "usd",
    "hackerOneBountyAmount": 250,
    "hackerOneDisclosedAt": "2021-01-25T20:12:19.503Z",
    "hackerOneCreatedAt": "2020-03-07T13:51:40.910Z",
    "hackerOneReporter": "makerlab",
    "hackerOneTitle": "Denial of Service by requesting to reset a password",
    "nvdUrl": "https://nvd.nist.gov/vuln/detail/CVE-2020-8295",
    "nvdWeaknesses": "CWE-400",
    "nvdDescription": "A wrong check in Nextcloud Server 19 and prior allowed to perform a denial of service attack when resetting the password for a user.",
    "nvdMetric": "cvssV3",
    "nvdBaseScore": 7.5,
    "nvdBaseSeverity": "HIGH",
    "nvdPublishedData": "2021-01-26T18:16Z",
    "nvdLastModifiedDate": "2021-02-02T21:02Z",
    "cpe": "cpe:2.3:a:nextcloud:nextcloud_server:*:*:*:*:*:*:*:*"
}
```


Figure 4-23: A partial ABox view of the initial HackerOne system ontology

Figure 4-23 shows an example of a vulnerability that is found in both NVD and HackerOne for the NextCloud[38] example. Figure 4-24 shows a diagram with a partial list of vulnerabilities found for NextCloud and the resources each vulnerability is captured.

[38] https://nextcloud.com/

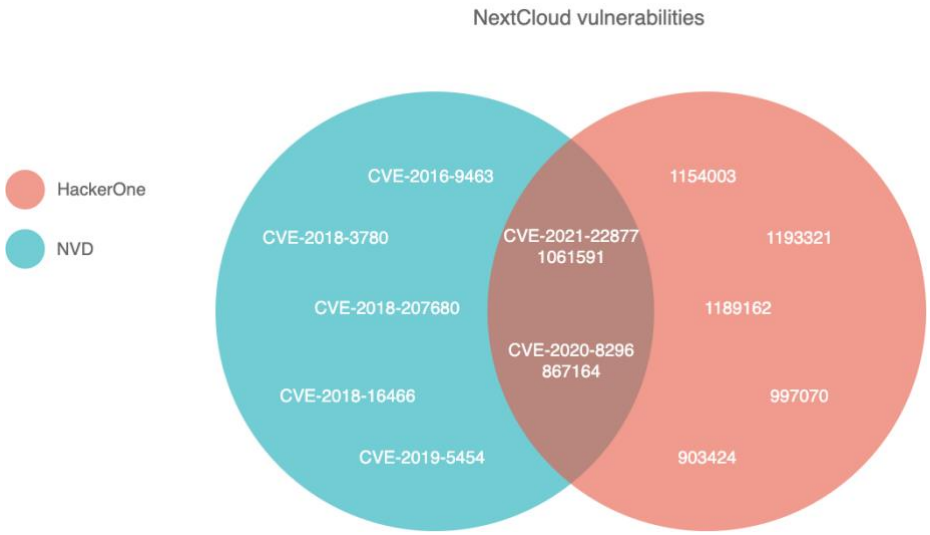

Figure 4-24: Nextcloud reports in NVD and HackerOne

While NVD follows a standardized convention to specify applicable platforms using structured naming scheme provided by CPE, HackerOne and other bug bounty platforms do not necessarily adhere to this standardized name scheme for products and domains of products/environments for which vulnerabilities have been reported. Due to these non-standardized descriptions used by bounty platforms, the content and level of detail provided in hacktivity reports can vary not only across bounty platforms, but also within the same bounty platform. Due to these inconsistencies, links established between these ontologies using the software product/environment or reporter name will be of low quality in terms of their recall and precision.

Another issue related to data accuracy is the lack of versioning information in HackerOne. Each vulnerability is applicable on a list of the application versions until a patch is introduced and the vulnerability is resolved in one update. We attempted to mitigate this problem by using the report date of a vulnerability submission to evaluate the threat rather than the vulnerable versions. It should be noted that these links will be of lower quality. While recall might be acceptable for these links, their precision might be low. This is since we no longer distinguish the specific version

of a product in which a vulnerability might have been reported and only consider the product version closest to the hacktivity reporting date. The link quality could be further improved, by including additional resources in the linking process, such as references made to hacktivity reports in commit messages. Figure 4-25 shows a partial view of how different concepts are connected to each other, while each can function independently through their own ontologies.

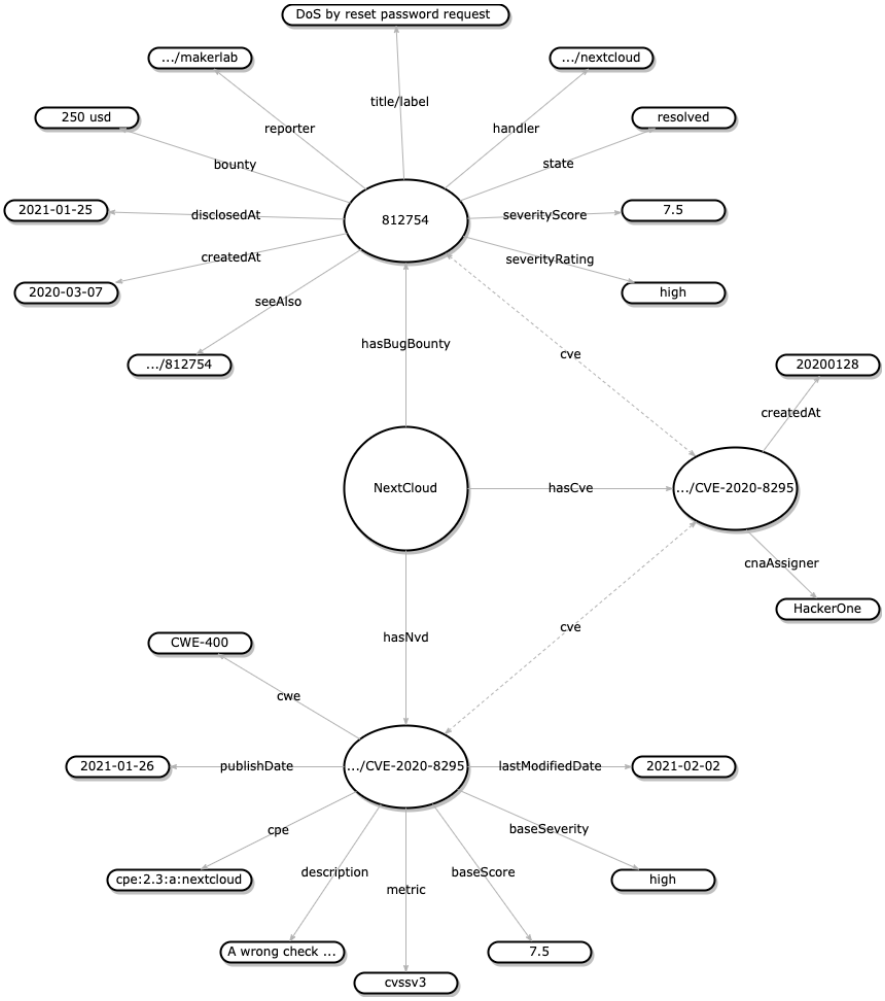

Figure 4-25: Nextcloud graph example

4.3.4 External Integrations – DBpedia

The DBpedia Knowledge Graph encompasses curated data from Wikipedia and allows for the integration of Wikipedia data with other knowledge graphs using SPARQL. In our research, this integration can take on two forms. 1) We can use DBpedia data to further enrich resources in our ontologies, and 2) specify concepts through annotated text properties using DBpedia Spotlight [66]. DBpedia Spotlight allows for the automatic annotating mentions of DBpedia resources in text. This allows for linking unstructured information sources to the Linked Open Data cloud [67]. Figure 4-26 illustrates how we establish the link between DBpedia and CWE using the same-as relationship to describe each CWE property.

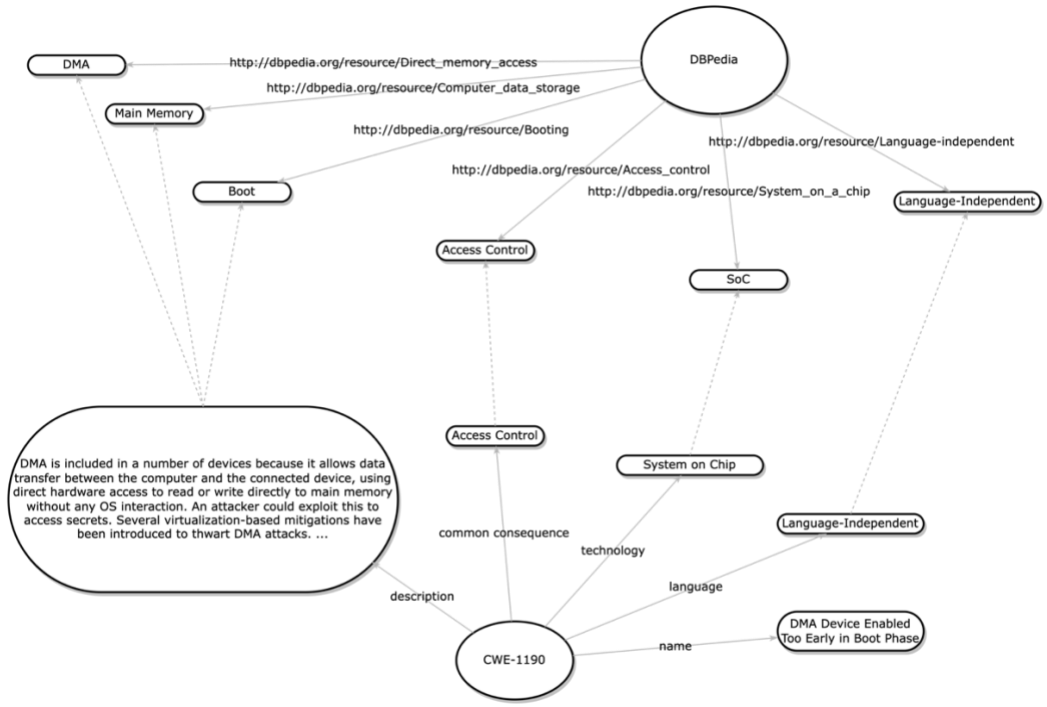

Figure 4-26: Linking DBpedia to CWE.

4.3.5 Google Knowledge Graph

The most widely used approach to identify and access knowledge resources found on the Internet are Search Engines. The Google Knowledge Graph is a knowledge base from which Google serves relevant information beside its search results. This information allows users to see answer in a glance. The data used by the Google Knowledge Graphs [68] is generated automatically from a variety of sources, covering people, businesses, places and more. We tried to integrate the Google knowledge graph with our own knowledge graph, however, we soon realized that the Google Knowledge Graph is used internally by Google to enrich their search results. Access to the Google KG is limited to a REST API, which accepts a few parameters as query terms and returns the result as a list of ranked objects. While this traditional API is commonly used for Web-based applications, this API does not permit a full integration of the Google Knowledge Graph with our knowledge graph. More specifically, without exposing the internal model of their KG nor providing a SPARQL endpoint, no semantic links between the Google knowledge graph and our knowledge graphs can be established. As a result, we can not seamlessly semantic inference can be inducted across resource boundaries. We were therefore unable to directly integrate the Google knowledge graph with our knowledge graph.

Chapter 5

Use Cases

In this chapter, we illustrate how our modeling approach can support different types of software analysis services (scenarios). More specifically, these analytic services are based on user-defined queries that take advantage of both, our knowledge graph and SW inference services. For the first analysis service example (Section 5.2), we integrate knowledge from build management systems (e.g., Maven) with knowledge from vulnerability-related resources (e.g., HackerOne) to identify potentially vulnerable system components that might be affected by a vulnerability reported in a bounty program (HackerOne). For the second use case (Section 5.3), we use our KG to analyze not only how many hacktivity reports a bounty hunter has submitted, but also the impact of these detected vulnerabilities on a software ecosystem. For the third use case (Section 5.4), we classify a bounty hunter's expertise based on their hacktivity reports and use this information to identify a bounty hunter's main expertise areas. For the last use case (Section 5.5), we show how our approach can be seamlessly integrated with other already existing knowledge graphs (e.g., DBpedia) to provide an enriched knowledge base.

5.1 Case Studies Setup

For our case studies' data collection and extraction, we rely on five real-world data sources: NVD, Dependencies, HackerOne, Snyk, and CWE. For our study, we limited the dataset to projects for

which we could identify the CVE-ID in both the bounty program reports and NVD. The extracted facts are then populated in their corresponding ontologies making them persistent in our triple store.

Table 5-1 provides an overview of the dataset which we created and used to populate our knowledge graph. The table shows the TBox (concepts and properties) of our ontologies. It should be noted that the size of the serialized ABox in our triplestore is 203,390 explicit triples and 22,385,063 inferred.

Concept	TBox size	Description
CVE/NVD	841	A subset of NVD records with HackerOne or Snyk reference
Snyk	391	A subset of Snyk records that are referenced in NVD
CWE	941	All records in CWE v4.4
CPE	239	A subset of CPE records for used products only
HackerOne Reports	450	A subset of HackerOne records that are referenced in NVD
HackerOne Hackers	247	Number of Hackers for selected HackerOne Reports
Libraries	5908	A subset of used libraries with their dependencies

Table 5-1: Case studies environment.

5.2 Dependency graph vulnerabilities

As discussed earlier, while programmers are often unaware of components on which their system indirectly depends on, these indirect dependencies still need to be considered a part of a system's attack surface and therefore closely monitored. In this case study, we illustrate how our KG in combination with SW inference services can be used to identify vulnerable components (**Error! Reference source not found.**). The analysis is based on a SPARQL query that identifies s ystem components that might be directly or indirectly affected by a vulnerability reported in HackerOne. More specifically, the analysis introduced in the case study considers both direct and indirect build dependencies extracted from the dependency management system and combines this

information with CVE and CPE vulnerabilities to identify components for which a vulnerability report in the bounty program (HackerOne) exists.

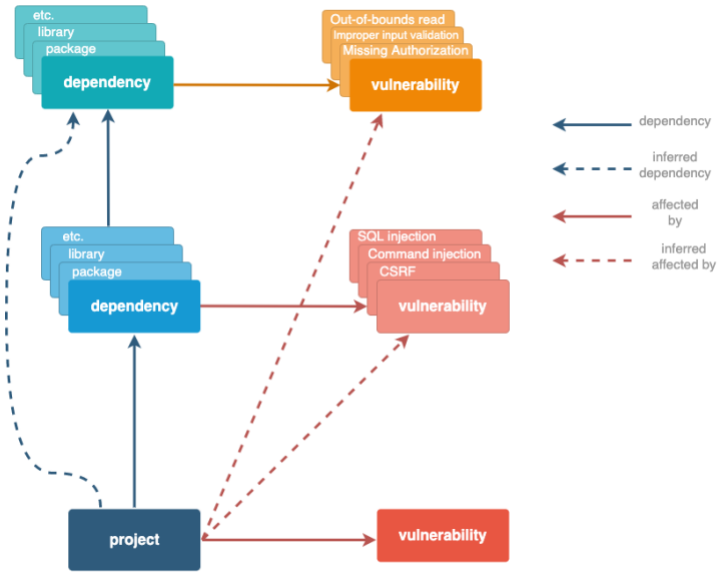

Figure 5-1: Dependency graph vulnerability overview

The project *mikel/mail*[39] is a library for Ruby that is designed to handle email generation, parsing, and sending. This repository has 3500 stars with 228 contributors and is used by 1.9 million repositories on GitHub only. The following example demonstrates recorded or inferred vulnerabilities of this system, which has a total of 5 direct and 3402 indirect dependencies to other components. It should be noted that while there are many indirect dependencies that are used in a project, this does not necessarily mean the project has inherited a vulnerability from these dependencies. Thus, this assertion can only be proved through a detailed analysis of the source code, which is out of the scope of this thesis. In Figure 5-2 we presented a partial view of the projects that our query scans for any potential vulnerabilities in *mikel/mail* (direct dependencies are shown in solid lines, indirect dependencies are in dashed lines).

[39] https://github.com/mikel/mail

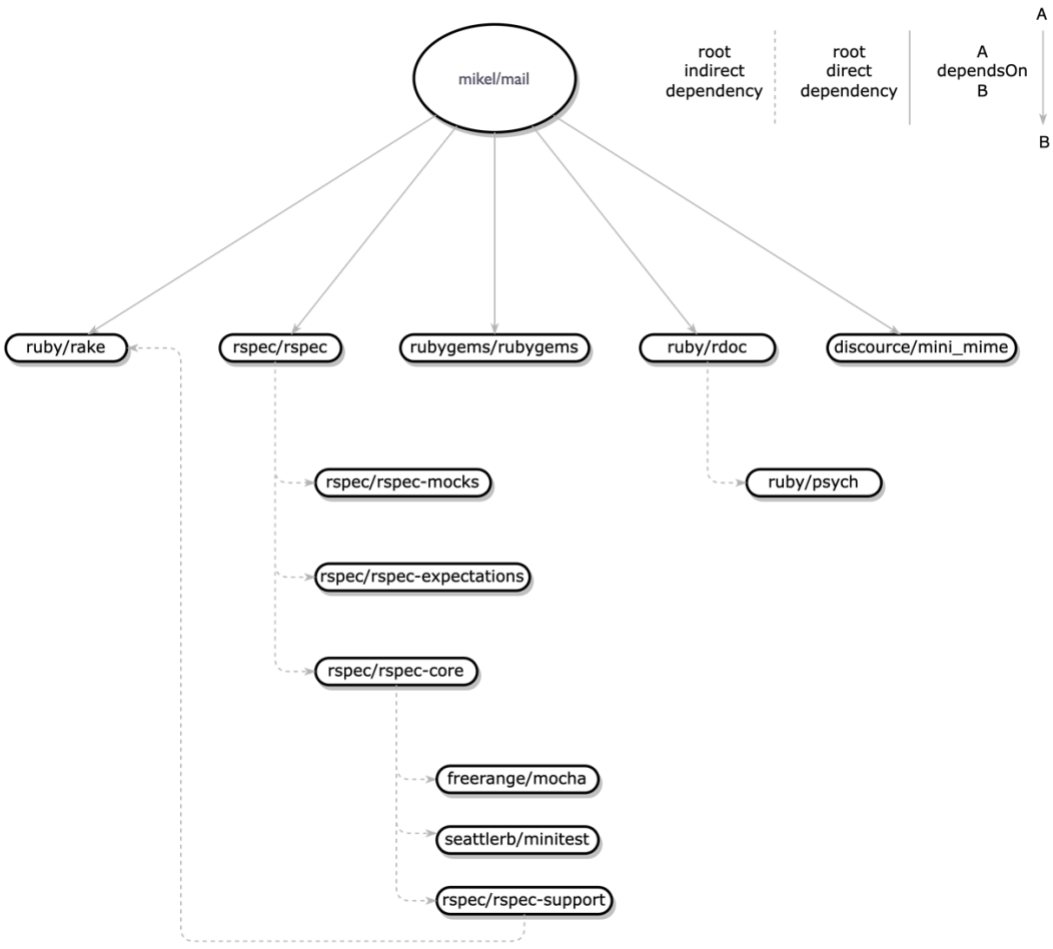

Figure 5-2: Partial dependency graph for "mikel/mail "

The query shown in Figure 5-3 retrieves all HackerOne reports for any component *mikel/mail* directly depends on. The results (in **Error! Reference source not found.**) show that o nly one direct dependent component has a HackerOne report with a known vulnerability associated with.

```
PREFIX hackerones: <http://encs.concordia.ca/ontologies/2021/hackerone#>
PREFIX nvds: <http://encs.concordia.ca/ontologies/2021/nvd#>
PREFIX sv: <http://encs.concordia.ca/ontologies/2021/software-vulnerability#>
PREFIX mitres: <http://encs.concordia.ca/ontologies/2021/mitre#>
PREFIX nvd: <http://encs.concordia.ca/ontologies/2021/nvd/resource/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select ?product ?hOneReport ?cweName ?severity ?state ?disclosedDate where {
    ?lib sv:repositoryUrl "https://github.com/mikel/mail"^^xsd:anyURI.
    ?lib sv:libCpes ?cpes.
    ?cpes nvds:hasVulnerabilities ?nvds.
    ?nvds mitres:Cve ?cves.
    ?hOne a hackerones:HackerOneVul.
    ?hOne mitres:Cve ?cves.
    ?hOne rdfs:label ?hOneReport.
    ?cpes mitres:cpeProduct ?product.
    ?nvds mitres:Cwe ?cwes.
    ?cwes mitres:Name ?cweName.
    ?hOne hackerones:disclosedDate ?disclosedDate.
    OPTIONAL { ?hOne hackerones:severity ?severity }.
    OPTIONAL { ?hOne hackerones:substate ?state }.
}
ORDER BY DESC(?disclosedDate)
```

Figure 5-3: Direct bounty query for "mikel/mail" project

	product	hOneReport	cweName	severity	state	disclosedDate
1	"mail"	"SMTP command injection"	"Improper Neutralization of CRLF Sequences ('CRLF Injection')"		"resolved"	"2016-06-30T07:28:29.785000+00:00"^^xsd:dateTime

Figure 5-4: Direct bounty result for "mikel/mail" project

In what follows, we extend our previous analysis (Figure 5-3) to additionally include indirect dependencies in the analysis. The refined query, shown in Figure 5-5, considers transitive (indirect) dependencies by inferring the transitive properties captured within our KG. Including both direct and indirect dependencies shows that there are only 19 additional indirect dependent components (out of 3402) in the *mikel/mail* system that have vulnerabilities reported in HackerOne. It should be noted that the dependency analysis provided by this query does not consider the actual code usage of the component code. Therefore, our analysis may report that a system is exposed to a vulnerable component, even if the vulnerable part of the component is not used and therefore might not affect the system. A partial result of the transitive query for *mikel/mail* is provided in Figure 5-6.

```
PREFIX hackerones: <http://encs.concordia.ca/ontologies/2021/hackerone#>
PREFIX nvds: <http://encs.concordia.ca/ontologies/2021/nvd#>
PREFIX sv: <http://encs.concordia.ca/ontologies/2021/software-vulnerability#>
PREFIX mitres: <http://encs.concordia.ca/ontologies/2021/mitre#>
PREFIX nvd: <http://encs.concordia.ca/ontologies/2021/nvd/resource/>
select ?depLib ?product ?hOneReport ?cweName ?severity ?state ?disclosedDate where {
    ?lib sv:repositoryUrl "https://github.com/mikel/mail"^^xsd:anyURI.
    {
        ?lib sv:dependency ?depLib.
        ?depLib sv:libCpes ?cpes.
        ?cpes nvds:hasVulnerabilities ?nvds.
        ?nvds mitres:Cve ?cves.
        ?hOne a hackerones:HackerOneVul.
        ?hOne mitres:Cve ?cves.
    }
    ?hOne rdfs:label ?hOneReport.
    ?cpes mitres:cpeProduct ?product.
    ?nvds mitres:Cwe ?cwes.
    ?cwes mitres:Name ?cweName.
    ?hOne hackerones:disclosedDate ?disclosedDate.
    OPTIONAL { ?hOne hackerones:severity ?severity }.
    OPTIONAL { ?hOne hackerones:substate ?state }.
}
ORDER BY DESC(?disclosedDate)
```

Figure 5-5: Transitive bounty query for "mikel/mail"

	depLib	product	hOneReport	cweName	severity	state	disclosedDate
1	libs:github.com/rvagg/bl	"bufferlist"	"[bl] Uninitialized memory exposure via negative .consume()"	"Out-of-bounds Read"	"high"	"resolved"	"2020-08-27T15:16:42.547000+00:00"^^xsd:dateTime
2	libs:github.com/zaach/jison	"jison"	"OS Command Injection on Jison [all-parser-ports]"	"Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')"	"medium"	"resolved"	"2020-05-28T10:39:50.786000+00:00"^^xsd:dateTime
3	libs:github.com/yarnpkg/yarn	"yarn"	"[yarn] yarn.lock integrity & hash check logic is broken"	"Time-of-check Time-of-use (TOCTOU) Race Condition"	"critical"	"resolved"	"2020-02-26T13:46:41.721000+00:00"^^xsd:dateTime
4	libs:github.com/yarnpkg/yarn	"yarn"	"Filesystem Writes via `yarn install` via symlinks and tar transforms inside a crafted malicious package"	"Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')"	"medium"	"resolved"	"2020-02-15T00:33:41.258000+00:00"^^xsd:dateTime
5	libs:github.com/unshiftio/url-parse	"url-parse"	"[url-parse] Improper Validation and Sanitization"	"Improper Input Validation"	"high"	"resolved"	"2020-01-27T09:10:53.941000+00:00"^^xsd:dateTime
6	libs:github.com/ruby/rake	"rake"	"OS Command Injection via egrep in Rake::FileList"	"Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')"		"resolved"	"2019-08-29T03:12:56.050000+00:00"^^xsd:dateTime

Figure 5-6: Partial result of the transitive query for "mikel/mail"

5.3 Impact of Bounty Hunters

In this case study, we illustrate how using our KG and a SPARQL query, one can identify bounty hunters and the impact of the discovered vulnerabilities on global software ecosystem. More specifically, in this analysis, we not only analyze how many hacktivity reports a bounty hunter has submitted but also the potential impact of these detected vulnerabilities on the software ecosystem. This information can be used to identify and rank bounty hunters based on the potential impact of their work on global software community.

```
PREFIX hackerones: <http://encs.concordia.ca/ontologies/2021/hackerone#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX mitres: <http://encs.concordia.ca/ontologies/2021/mitre#>
PREFIX nvds: <http://encs.concordia.ca/ontologies/2021/nvd#>
PREFIX sv: <http://encs.concordia.ca/ontologies/2021/software-vulnerabili

select (COUNT(*) as ?Triples) where {
    ?p a hackerones:PenetrationTester .
    ?p foaf:name "Rafal Janicki".
    ?user hackerones:hasPerson ?p.
    ?user hackerones:reports ?reports.
    ?reports mitres:Cve ?cves.
    ?nvds mitres:Cve ?cves.
    ?nvds a nvds:NvdVul.
    ?nvds nvds:hasCpe ?cpes.
    ?cpes mitres:parentLib ?libs.
}
```

Figure 5-7: Direct contribution measurement query for "Rafal Janicki"

	Triples
1	"6"^^xsd:integer

Figure 5-8: Direct contribution measurements result for "Rafal Janicki"

Figure 5-77 and Figure 5-8 show the query and its results for bug bounty hunter *"Rafal Janicki",* who has submitted a total of 6 bug bounty hacktivity reports. For this query, we only consider the components that are directly dependent on the vulnerability. The query shown in Figure 5-9 extends the analysis to include also components/projects that are directly or indirectly

dependent on the reported vulnerabilities. This extended analysis shows that a total of 1899 libraries (refer to **Error! Reference source not found.**) are potentially affected by the v ulnerabilities reported by this bounty hunter. The query highlights the ability of our modeling approach not only to integrate resources (by establishing links between the bug bounty program, NVD and Maven), but also to infer new knowledge. More specifically, the query takes advantage of the bi-directional links we established in our build management KG that allows us to infer a global dependency graph (using the transitive dependencies). Furthermore, our analysis can be easily extended to include additional bounty programs and/or build management systems.

```
PREFIX hackerones: <http://encs.concordia.ca/ontologies/2021/hackerone#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX mitres: <http://encs.concordia.ca/ontologies/2021/mitre#>
PREFIX nvds: <http://encs.concordia.ca/ontologies/2021/nvd#>
PREFIX sv: <http://encs.concordia.ca/ontologies/2021/software-vulnerability#

select (COUNT(*) as ?Triples) where {
    ?p a hackerones:PenetrationTester .
    ?p foaf:name "Rafal Janicki".
    ?user hackerones:hasPerson ?p.
    ?user hackerones:reports ?reports.
    ?reports mitres:Cve ?cves.
    ?nvds mitres:Cve ?cves.
    ?nvds a nvds:NvdVul.
    ?nvds nvds:hasCpe ?cpes.
    ?cpes mitres:parentLib ?libs.
    {
    }
    UNION
    {
        ?libs sv:dependent ?deps.
    }
}
```

Figure 5-9: Indirect contribution measurements query for "Rafal Janicki"

	Triples
1	"1899"^^xsd:integer

Figure 5-10: Indirect contribution measurements result for "Rafal Janicki"

5.4 Classification of Bounty Hunter's Expertise

In this use case, we construct a query that classifies a bounty hunter's expertise for a certain type of vulnerabilities by combining the information from the CWE and the submitted hacktivity reports by a bounty hunter. The weaknesses are derived from the CWE repository by connecting the hacktivity reports to the CVE (NVD) and then to the CWE repository. This analysis could be further extended to include also Common Attack Pattern Enumeration and Classification (CAPEC), or other related vulnerability classification information. For instance, a bounty hunter might have discovered vulnerabilities of type weakness "Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')" more frequently than any other types of weaknesses. The result of such a classification/ranking can be used to identify bounty hunters with expertise in certain areas.

```
PREFIX hackerones: <http://encs.concordia.ca/ontologies/2021/hackerone#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX mitres: <http://encs.concordia.ca/ontologies/2021/mitre#>
PREFIX nvds: <http://encs.concordia.ca/ontologies/2021/nvd#>
select (COUNT(*) as ?numberOfReports) ?cweId ?cweName where {
    ?person foaf:name "Rafal Janicki".
    ?person a hackerones:PenetrationTester .
    ?hOneUser hackerones:hasPerson ?person.
    ?hOneUser hackerones:reports ?reports.
    ?reports mitres:Cve ?cve.
    ?nvd mitres:Cve ?cve.
    ?nvd a nvds:NvdVul.
    ?nvd mitres:Cwe ?cwe.
    ?cwe mitres:Name ?cweName.
    ?cwe mitres:Id ?cweId.
}
group by ?cweId ?cweName
order by DESC(?numberOfReports)
```

Figure 5-11: Expertise assessment query for "Rafal Janicki"

	numberOfReports	cweId	cweName
1	"15"^^xsd:integer	"22"	"Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')"
2	"12"^^xsd:integer	"79"	"Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')"
3	"1"^^xsd:integer	"89"	"Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')"

Figure 5-12: Expertise assessment result for "Rafal Janicki"

The query in Figure 5-11 retrieves all hacktivity reports published by Rafael Janicki on HackerOne and then classifies these based on their reported vulnerability types, using information from NVD and CWE. As the query results (Figure 5-122) show, the bounty hunter has mostly worked on vulnerabilities of CWE types "22" ("Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')") and "79" (Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')) of a Pathname to a Restricted Directory ('Path Traversal').

5.5 Integration With External KGs

The following use case presents, how approach can be further enriched with knowledge from external (third-party) knowledge graphs. The query in Figure 5-13 shows how one can create a profile for a software product, in this example for *Nextcloud*. The query, firstly retrieves general information (e.g., name, developer, logo, OS, product description) from DBpedia and secondly, combines this with general information with knowledge extracted in our KG (hacktivity reports and NVD information).

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX hackerones: <http://encs.concordia.ca/ontologies/2021/hackerone#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select *
WHERE {
    {
        SERVICE <https://dbpedia.org/sparql/> {
            SELECT ?property ?object
            {
                <http://dbpedia.org/resource/Nextcloud> ?property ?object
                filter(langMatches(lang(?object),"en"))
            }
        }
    }
    UNION
    {
        SERVICE <http://localhost:7200/repositories/vdb> {
            select ?title ?publishDate ?severity ?bountyAmount where {
                ?hOne a hackerones:HackerOneVul.
                ?hOne hackerones:handler "nextcloud".
                ?hOne rdfs:label ?title.
                ?hOne hackerones:publishedDate ?publishDate.
                OPTIONAL {
                    ?hOne hackerones:severity ?severity.
                    ?hOne hackerones:bountyAmount ?bountyAmount.
                }
            }
        }
    }
}
```

Figure 5-13: DBpedia integration query

The resulting profile includes a total of 111 facts, where 11 facts are extracted from the external DBpedia KG (Figure 5-14) and the remaining facts are extracted from our KG (Figure 5-15).

	property	object
1	rdfs:label	"Nextcloud"@en
2	rdfs:comment	"Nextcloud is a suite of client-server software for creating and using file hosting services. It is enterprise-ready with comprehensive support options. Being free and open-source software, anyone is allowed to install and operate it on their own private server devices. The original ownCloud developer Frank Karlitschek forked ownCloud and created Nextcloud, which continues to be actively developed by Karlitschek and other members of the original ownCloud team."@en
3	ns1:name	"Nextcloud"@en
4	http://dbpedia.org/property/name	"Nextcloud"@en
5	http://dbpedia.org/ontology/abstract	"Nextcloud is a suite of client-server software for creating and using file hosting services. It is enterprise-ready with comprehensive support options. Being free and open-source software, anyone is allowed to install and operate it on their own private server devices. Nextcloud is functionally similar to Dropbox, Office 365 or Google Drive when used with its integrated office suite solutions Collabora Online or OnlyOffice. It can be hosted in the cloud or on-premises. It is scalable from home office solutions based on the low cost Raspberry Pi all the way through to full sized data centre solutions that support millions of users. The original ownCloud developer Frank Karlitschek forked ownCloud and created Nextcloud, which continues to be actively developed by Karlitschek and other members of the original ownCloud team."@en
6	http://dbpedia.org/property/screenshotAlt	"Nextcloud Talk"@en
7	http://dbpedia.org/property/developer	"Nextcloud GmbH., Community"@en
8	http://dbpedia.org/property/logo	"Nextcloud Logo.svg"@en
9	http://dbpedia.org/property/operatingSyst	"Server: Linux"@en
10	http://dbpedia.org/property/operatingSyst	"Clients: Windows, macOS, Linux, Android, iOS"@en

Figure 5-14: A partial view DBpedia integration result - part 1

title	publishDate	severity	bountyAmount
"Potential DDoS when posting long data into workflow validation rules"	"2020-10-25T04:00:29.184000+00:00"^^xsd:dateTime	"medium"	"100"
"Stored XSS in markdown file with Nextcloud Talk using Internet Explorer"	"2020-11-01T10:41:13.962000+00:00"^^xsd:dateTime	"low"	"150"
"Acting under any different user via DB-stored credentials"	"2020-12-18T12:53:55.604000+00:00"^^xsd:dateTime	"high"	"250"
"Secure View" aka "Hide Download" can be bypassed easily	"2020-02-03T13:18:19.732000+00:00"^^xsd:dateTime	"high"	"100"

Figure 5-15: A partial view DBpedia integration result - part 2

Chapter 6

Related Work and Discussion

In this chapter, we discuss relevant to our work and how our contributions related to the existing research.

6.1 Modeling known vulnerabilities with Ontologies

In what follows, we discuss the use of ontologies as a modeling language in the cybersecurity domain. Cybersecurity is concerned with technologies, processes, and practices designed to protect networks, computers, programs, and data from attacks, damages, or unauthorized access [68]. More specifically, we focus on work related to the modeling known vulnerabilities using ontologies. Undercoffer et al. [69] and [70] introduced an ontology for intrusion detection systems. The proposed ontology was created based on the evaluation of 4000 vulnerabilities and the attack strategies used to exploit them. Their ontology was specified using the DARPA Agent Markup Language (DAML) and prototyped using DAMLJessKB [71]. The authors included several use case scenarios based on common attacks such as Denial of Service – Sync Flood, the Classic Mitnick Type Attack, and Buffer Overflow Attack.

In More et al. [72], a situation-aware intrusion detection model was presented that integrates system security data sources (e.g., network logs) to create a semantically rich knowledge base for the detection of cyber threats/vulnerabilities. The authors collected data streams from the network monitors, host monitors, sensor data, and other Intrusion Detection Systems (IDS)

modules, which are asserted as facts in their knowledge base (introduced by [69] and [70]). For intrusion detection, the authors take the advantage of SW reasoning services to infer whether there is an indication of an attack.

Joshi et al. [73] extracted cybersecurity-related entities, concepts, and relations in their work, and captured them in their IDS ontology (introduced by [69] and [70], and further enhanced by [72]). As a part of their approach, the authors also mapped these concepts to the objects in the DBpedia knowledge base using DBpedia Spotlight [67]. The approach creates an RDF-linked data representation of cybersecurity concepts and vulnerability descriptions. The security information is extracted from both structured vulnerability databases and unstructured text. Their approach supports vulnerability identification and vulnerability mitigation efforts.

Sartabanova et al. [74] proposed an ontology for CWE and software architecture. The authors used Protégé to model the weakness structure and design their ontology. In an effort by Iannacone et al. [75], the existing cybersecurity ontologies from [69], [70], and [76] were extended to introduce an ontology framework that incorporates additional information from a variety of structured and unstructured data sources. In the study of Kamongi et al. [77], the authors introduced VULCAN, a vulnerability assessment framework for cloud computing systems. The framework consists of two main components: an Ontological Vulnerability Assessment introduced by [78] and an Ontology Vulnerability Database [79] component. These two components provide access to known vulnerability information that has been published by NVD. For the vulnerability assessment, their approach benefits from advanced reasoning capabilities to support a semantic search for vulnerabilities.

Like most of the related research on modeling vulnerabilities with ontologies, our research describes vulnerability information. We reuse existing ontologies [10] and extend these already available ontologies with new vulnerability information (concepts), as well as introduce new ontologies that capture knowledge specific to the bug bounty domain. None of the existing research has included the domain of bounty programs in its models.

6.2 Knowledge Integration

Very little research work exists that discusses the integration of vulnerability-related knowledge with other software artifacts using ontologies. The closest work related to ours is the work by Alqahtani et al. [65] who introduced SEVONT, a unified ontology for modeling software vulnerabilities. SEVONT introduces a multi-layer knowledge model which not only provides a unified knowledge representation, but also captures software vulnerability information at different abstract levels to allow seamless integration, analysis, and reuse of the modeled knowledge. Furthermore, Eghan et al. [80] developed a new ontology to link bi-directional dependencies in Maven using OWL reasoning service which was also integrated with [52]. Syed et al. [76], introduced Unified Cybersecurity Ontology (UCO) to support information integration and cyber situational awareness. The ontology integrates data and knowledge schema from both cybersecurity systems and commonly used cybersecurity standards to allow for data exchange among these resources. The UCO ontology has also been mapped to a number of other existing cybersecurity ontologies ([69] and [70]) and resources on the Linked Open Data cloud ([72] and [73]). As part of their work, the authors presented a unified representation model for several vulnerability databases to facilitate data retrievals from different data sources.

Our ontology is based on SEVONT, however, we contributed by extending the framework and including ontologies related to the bug bounty domain. The resulting unified representation of software artifacts related to known software vulnerabilities provides the foundation for novel software analytics services which include the use of information captured in bounty programs.

6.3 Ontology-based Analytics tools that support Vulnerability Analysis

In the research domain of ontology-based analytics tools, Gyrard et al. [81] and [82] contributed by introducing an ontology-based Security Toolbox for Attack and Countermeasure (STAC). STAC guides software developers in selecting the appropriate security mechanisms to secure Internet of Things (IoT) applications (more specifically, securing ETSI Machine to Machine [M2M] architecture). Moreover, Durai et al. [83] proposed an ontology (SQLIO) that can be used to prevent and detect SQL Injection Attacks (SQLIA). The objective of the methodology is to prevent and detect SQLIA web vulnerabilities in cloud environments. Zhang et al. [84] conducted an empirical study about applying data-mining techniques on NVD data with the objective of predicting the time for the occurrence of a new vulnerability in a given software application. They experimented with different features extracted from the information available in NVD and applied various machine-learning algorithms to examine the predictive power of the data and features. Their results show that the data in NVD generally have mostly poor prediction capability, except for a few vendors and software applications.

Like other existing research, we do provide software analytics support to manage known software vulnerabilities. However, our approach differs from existing approaches, by taking advantage of SW inference services to include knowledge from the bug bounty domain. This additional knowledge can provide developers with new insights on the impact and management of vulnerabilities. We furthermore illustrate how more generic knowledge resources such as DBpedia can be included to further enrich our vulnerability analysis results.

Chapter 7

Conclusions and Future Work

In this thesis, we introduce a knowledge graph that takes advantage of the SW and its technology stack to leverage and integrate knowledge resources related to known software vulnerabilities. More specifically, we utilized knowledge graphs to provide a unified representation that allows us to transform heterogeneous information silos (e.g., VDBs, bug bounty programs, build systems, and source code repositories) into information hubs. We introduced a methodology to depict the major steps involved in identifying, modeling, and integrating knowledge resources to form such a knowledge graph. We then illustrated how our methodology can be applied to create a knowledge graph for resources related to known software vulnerabilities. For our knowledge modeling approach, we reuse existing ontologies (e.g., SEVONT, SBSON) and enrich the knowledge graph with new resources (e.g., bounty programs, DBpedia). This enriched knowledge graph serves an information hub that enables us to infer knowledge not only within but also across knowledge resource boundaries. We introduce novel software security analytics services that take advantage of our unified knowledge representation.

At last, we illustrate the applicability of our approach through several case studies, which include real-world data sources such as: NVD-CVE, Maven, HackerOne, Snyk, and CWE.

There are also several possible directions for future work:

Tool development. One potential avenue for future work is the integration of our knowledge graph within an IDE. More specifically, one could envision taking advantage of a programmer's current task context in the IDE (e.g., current source code being edited) to provide context-specific analytics services.

Improve ontology linking. While we rely on the exact concept and property matching (e.g., CVE-IDs) techniques to link ontologies, this limits the number of instances that we can link. Moreover, lack of investigating the actual usage of vulnerable code increased the number of project dependencies exponentially. Additional semantic linking techniques could be applied to improve recall while linking our ontologies.

Integrating additional resources

Another future direction would be to integrate other software-related knowledge resources, such as StackOverflow[40], to further enrich our knowledge graph.

[40] https://stackoverflow.com/

References

[1] K. R. van Wyk and G. McGraw, "Bridging the gap between software development and information security," *IEEE Security & Privacy,* vol. 3, no. 5, pp. 75-79, 2005.

[2] A. L. Mesquida and A. Mas, "Implementing information security best practices on software lifecycle processes: The ISO/IEC 15504 Security Extension," *Computers & Security,* vol. 48, pp. 19-34, 2015.

[3] T. Walshe and A. Simpson, "An Empirical Study of Bug Bounty Programs," in *IBF 2020 - Proceedings of the 2020 IEEE 2nd International Workshop on Intelligent Bug Fixing*, 2020, doi: 10.1109/IBF50092.2020.9034828.

[4] H. Booth, D. Rike, and G. A. Witte, "The national vulnerability database (nvd): Overview," in *ITL BULLETIN FOR DECEMBER 2013*, ed, 2013.

[5] DeveloperSteve, "Using the Snyk Vulnerability Database to find projects for The Big Fix," vol. July, 2022, ed. Snyk.io: Snyk.io, 2022.

[6] C. Theisen, N. Munaiah, M. Al-Zyoud, J. C. Carver, A. Meneely, and L. Williams, "Attack surface definitions: A systematic literature review," *Information and Software Technology,* vol. 104, pp. 94-103, 2018.

[7] S. S. Alqahtani, E. E. Eghan, and J. Rilling, "Tracing known security vulnerabilities in software repositories–A Semantic Web enabled modeling approach," *Science of Computer Programming,* vol. 121, pp. 153-175, 2016.

[8] V. Lenarduzzi, A. Sillitti, and D. Taibi, "A survey on code analysis tools for software maintenance prediction," in *International Conference in Software Engineering for Defence Applications*, 2018: Springer, pp. 165-175.

[9] K. Goseva-Popstojanova and A. Perhinschi, "On the capability of static code analysis to detect security vulnerabilities," *Information and Software Technology,* vol. 68, pp. 18-33, 2015.

[10] S. S. Alqahtani, E. E. Eghan, and J. Rilling, "SV-AF—a security vulnerability analysis framework," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016: IEEE, pp. 219-229.

[11] E. E. Eghan and J. Rilling, "A Semantic Web-Enabled Approach for Dependency Management," *International Journal of Software Engineering and Knowledge Engineering,* vol. 32, no. 09, pp. 1307-1343, 2022.

[12] J. R. David Uberti, Catherine Stupp. "The Log4j Vulnerability: Millions of Attempts Made Per Hour to Exploit Software Flaw." Wall street journal. https://www.wsj.com/articles/what-is-the-log4j-vulnerability-11639446180 (accessed May, 2022.

[13] P. Paganini. "Quebec shuts down thousands of sites as disclosure of the Log4Shell flaw." https://securityaffairs.co/wordpress/125556/hacking/quebec-shut-down-sites-log4shell.html (accessed May, 2022.

[14] C. Sharma, "Tragedy of the Digital Commons," *North Carolina Law Review, Forthcoming,* p. 82, August , 2022.

[15] L. Tal. "The Log4j vulnerability and its impact on software supply chain security." Snyk. https://snyk.io/blog/log4j-vulnerability-software-supply-chain-security-log4shell/ (accessed June, 2022.

[16] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american,* vol. 284, no. 5, pp. 34-43, 2001.

[17] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition,* vol. 5, no. 2, pp. 199-220, 1993.

[18] S. Staab and R. Studer, *Handbook on ontologies*. Springer Science & Business Media, 2010.

[19] R. Laurini, "Pre-consensus ontologies and urban databases," in *Ontologies for Urban Development*: Springer, 2007, pp. 27-36.

[20] "OWL 2 Web Ontology Language Document Overview (Second Edition)." W3C. https://www.w3.org/TR/owl2-overview/ (accessed March, 2022.

[21] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, and D. Nardi, *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[22] V. Gezer and S. Bergweiler, "Service and Workflow Engineering based on Semantic Web Technologies," *The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies. International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM-16), located at Tenth International Conference on M,* no. October, 2016.

[23] "SPARQL Query Language for RDF." W3C. https://www.w3.org/2001/sw/wiki/SPARQL (accessed March, 2022.

[24] V. G. Castellana *et al.*, "Scaling RDF Triple Stores in Size and Performance. Modeling SPARQL Queries as Graph Homomorphism Routines," in *Handbook of Statistics*, vol. 33, 2015.

[25] A. Blumauer, *The Knowledge Graph Cookbook. Recipes that Work* (Twist). 2020.

[26] X. Chen, S. Jia, and Y. Xiang, "A review: Knowledge reasoning over knowledge graph," *Expert Systems with Applications,* vol. 141, p. 112948, 2020/03/01/ 2020, doi: https://doi.org/10.1016/j.eswa.2019.112948.

[27] S. Ji, S. Pan, E. Cambria, P. Marttinen, and S. Y. Philip, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 33, no. 2, pp. 494-514, 2021.

[28] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering,* vol. 29, no. 12, pp. 2724-2743, 2017.

[29] T. Berners-Lee. "Linked Data - Design Issues." W3C. https://www.w3.org/DesignIssues/LinkedData (accessed May, 2022.

[30] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data: The Story so Far," in *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, A. Sheth Ed. Hershey, PA, USA: IGI Global, 2011, pp. 205-227.

[31] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs," in *CEUR Workshop Proceedings*, 2016, vol. 1695.

[32] "Announcement: DBpedia Snapshot 2021-06 Release," in *DBpedia Blog* vol. 2022, ed: dbpedia, 2021.

[33] A. Singhal. "Introducing the Knowledge Graph: things, not strings." Google. https://blog.google/products/search/introducing-knowledge-graph-things-not/ (accessed March, 2022.

[34] J. Holze. "DBpedia Global: Data Beyond Wikipedia." dbpedia.org. https://www.dbpedia.org/blog/dbpedia-global/ (accessed July, 2022.

[35] M. Sharma and P. Soni, "Quantitative Analysis and Implementation of Relational and Graph Database Technologies," *International Journal of Modern Computer Science and Applications (IJMCSA) ISSN,* pp. 2321-2632, 2014.

[36] T. Nafees, N. Coull, I. Ferguson, and A. Sampson, "Vulnerability anti-patterns: a timeless way to capture poor software practices (vulnerabilities)," in *24th Conference on Pattern Languages of Programs*, 2018: The Hillside Group, p. 23.

[37] W. N. Robinson, S. D. Pawlowski, and V. Volkov, "Requirements interaction management," *ACM Comput. Surv.,* vol. 35, no. 2, pp. 132–190, 2003, doi: 10.1145/857076.857079.

[38] M. Isaac and S. Frenkel. "Facebook Security Breach Exposes Accounts of 50 Million Users." https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html (accessed June, 2022.

[39] W. Wang, F. Dumont, N. Niu, and G. Horton, "Detecting Software Security Vulnerabilities Via Requirements Dependency Analysis," *IEEE Transactions on Software Engineering,* vol. 48, no. 5, pp. 1665-1675, 2022, doi: 10.1109/TSE.2020.3030745.

[40] A. Anwar, A. Abusnaina, S. Chen, F. Li, and D. Mohaisen, "Cleaning the NVD: Comprehensive quality assessment, improvements, and analyses," *IEEE Transactions on Dependable and Secure Computing,* 2021.

[41] H. Fryer and E. Simperl, "Web science challenges in researching bug bounties," in *WebSci 2017 - Proceedings of the 2017 ACM Web Science Conference*, 2017, doi: 10.1145/3091478.3091517.

[42] "Hackers Earn Record-Breaking $100 Million on HackerOne." HackerOne. https://www.hackerone.com/press-release/hackers-earn-record-breaking-100-million-hackerone-0 (accessed June, 2022.

[43] K. Sridhar and M. Ng, "Hacking for good: Leveraging HackerOne data to develop an economic model of Bug Bounties," *Journal of Cybersecurity,* vol. 7, no. 1, p. tyab007, 2021.

[44] "Gartner's Top 25 Enterprise Software Startups To Watch In 2020." Forbes. https://www.forbes.com/sites/louiscolumbus/2020/07/05/gartners-top-25-enterprise-software-startups-to-watch-in-2020/?sh=5bab476d7822#486868977822 (accessed June, 2022.

[45] M. Linares-Vásquez, C. McMillan, D. Poshyvanyk, and M. Grechanik, "On using machine learning to automatically classify software applications into domain categories," *Empirical Software Engineering,* vol. 19, no. 3, pp. 582-618, 2014.

[46] O. Meqdadi, N. Alhindawi, J. Alsakran, A. Saifan, and H. Migdadi, "Mining software repositories for adaptive change commits using machine learning techniques," *Information and Software Technology,* vol. 109, pp. 80-91, 2019.

[47] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011: IEEE, pp. 133-142.

[48] N. Nazar, Y. Hu, and H. Jiang, "Summarizing software artifacts: A literature review," *Journal of Computer Science and Technology,* vol. 31, no. 5, pp. 883-909, 2016.

[49] S. Gupta and S. Gupta, "Summarization of software artifacts: a review," *International Journal of Computer Science & Information Technology (IJCSIT) Vol,* vol. 9, 2017.

[50] J. Zhang, R. Xie, W. Ye, Y. Zhang, and S. Zhang, "Exploiting code knowledge graph for bug localization via bi-directional attention," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 219-229.

[51] R. Xie *et al.*, "Deeplink: A code knowledge graph based deep learning approach for issue-commit link recovery," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019: IEEE, pp. 434-444.

[52] M. A. Musen, "The protégé project: a look back and a look forward," *AI matters,* vol. 1, no. 4, pp. 4-12, 2015.

[53] R. V. Guha, D. Brickley, and S. Macbeth, "Schema. org: evolution of structured data on the web," *Communications of the ACM,* vol. 59, no. 2, pp. 44-51, 2016.

[54] N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," ed: Stanford knowledge systems laboratory technical report KSL-01-05 and …, 2001.

[55] P. E. Van Der Vet and N. J. Mars, "Bottom-up construction of ontologies," *IEEE Transactions on Knowledge and data Engineering,* vol. 10, no. 4, pp. 513-526, 1998.

[56] V. Alves *et al.*, "An exploratory study of information retrieval techniques in domain analysis," in *2008 12th International Software Product Line Conference*, 2008: IEEE, pp. 67-76.

[57] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of software maintenance and evolution: Research and practice,* vol. 19, no. 2, pp. 77-131, 2007.

[58] K. K. Chaturvedi, V. Sing, and P. Singh, "Tools in mining software repositories," in *2013 13th International Conference on Computational Science and Its Applications*, 2013: IEEE, pp. 89-98.

[59] T. Siddiqui and A. Ahmad, "Data mining tools and techniques for mining software repositories: A systematic review," *Big Data Analytics,* pp. 717-726, 2018.

[60] J. Hejderup, A. van Deursen, and G. Gousios, "Software ecosystem call graph for dependency management," in *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, 2018: IEEE, pp. 101-104.

[61] R. Kikas, G. Gousios, M. Dumas, and D. Pfahl, "Structure and evolution of package dependency networks," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017: IEEE, pp. 102-112.

[62] "Mitre taps corporate partners to start up foundation focused on cyber defense," in *Washington Business Journal* vol. 2022, ed. https://www.bizjournals.com: Business Journal, 2019.

[63] E. E. Eghan, S. S. Alqahtani, C. Forbes, and J. Rilling, "API trustworthiness: an ontological approach for software library adoption," *Software Quality Journal,* vol. 27, no. 3, pp. 969-1014, 2019.

[64] M. Würsch, G. Ghezzi, M. Hert, G. Reif, and H. C. Gall, "SEON: A pyramid of ontologies for software evolution and its applications," *Computing,* vol. 94, no. 11, 2012, doi: 10.1007/s00607-012-0204-1.

[65] S. Alqahtani, "Enhancing Trust –A Unified Meta-Model for Software Security Vulnerability Analysis," ed. Montreal, 2018, pp. undefined-228.

[66] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes, "Improving efficiency and accuracy in multilingual entity extraction," in *Proceedings of the 9th international conference on semantic systems*, 2013, pp. 121-124.

[67] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer, "DBpedia spotlight: shedding light on the web of documents," presented at the Proceedings of the 7th International Conference on Semantic Systems, Graz, Austria, 2011. [Online]. Available: https://doi.org/10.1145/2063518.2063519.

[68] P. W. Singer and A. Friedman, *Cybersecurity: What everyone needs to know*. oup usa, 2014.

[69] J. Undercofer, A. Joshi, T. Finin, and J. Pinkston, "A target-centric ontology for intrusion detection," in *Workshop on Ontologies in Distributed Systems, held at The 18th International Joint Conference on Artificial Intelligence*, 2003.

[70] J. Undercoffer, A. Joshi, and J. Pinkston, "Modeling computer attacks: An ontology for intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*, 2003: Springer, pp. 113-135.

[71] J. B. Kopena and W. C. Regli, "DAMLJessKB: A tool for reasoning with the Semantic Web," in *International Semantic Web Conference*, 2003: Springer, pp. 628-643.

[72] S. More, M. Matthews, A. Joshi, and T. Finin, "A knowledge-based approach to intrusion detection modeling," in *2012 IEEE Symposium on Security and Privacy Workshops*, 2012: IEEE, pp. 75-81.

[73] A. Joshi, R. Lal, T. Finin, and A. Joshi, "Extracting cybersecurity related linked data from text," in *2013 IEEE Seventh International Conference on Semantic Computing*, 2013: IEEE, pp. 252-259.

[74] Z. Sartabanova, S. Sarsimbaeva, G. Urdabayeva, and V. Dimitrov, "Building an Ontology for CWE from the Point of View of Architectural Concept," in *CEUR Workshop Proceedings*, 2021, pp. 352-358.

[75] M. Iannacone *et al.*, "Developing an ontology for cyber security knowledge graphs," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, 2015, pp. 1-4.

[76] Z. Syed, A. Padia, T. Finin, L. Mathews, and A. Joshi, "UCO: A unified cybersecurity ontology," in *Workshops at the thirtieth AAAI conference on artificial intelligence*, 2016.

[77] P. Kamongi, S. Kotikela, K. Kavi, M. Gomathisankaran, and A. Singhal, "Vulcan: Vulnerability assessment framework for cloud computing," in *2013 IEEE 7th International Conference on Software Security and Reliability*, 2013: IEEE, pp. 218-226.

[78] A. Steele, "Ontological vulnerability assessment," in *International Conference on Web Information Systems Engineering*, 2008: Springer, pp. 24-35.

[79] S. Kotikela, K. Kavi, and M. Gomathisankaran, "Vulnerability assessment in cloud computing," in *Proceedings of the International Conference on Security and Management (SAM)*, 2012: The Steering Committee of The World Congress in Computer Science, Computer …, p. 1.

[80] E. E. Eghan, "Dependency Management 2.0–A Semantic Web Enabled Approach," Concordia University, 2019.

[81] A. Gyrard, C. Bonnet, and K. Boudaoud, "An ontology-based approach for helping to secure the etsi machine-to-machine architecture," in *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, 2014: IEEE, pp. 109-116.

[82] A. Gyrard, C. Bonnet, and K. Boudaoud, "The stac (security toolbox: attacks & countermeasures) ontology," in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 165-166.

[83] K. N. Durai, R. Subha, and A. Haldorai, "A Novel Method to Detect and Prevent SQLIA Using Ontology to Cloud Web Security," *Wireless Personal Communications,* vol. 117, no. 4, pp. 2995-3014, 2021.

[84] S. Zhang, X. Ou, and D. Caragea, "Predicting cyber risks through national vulnerability database," *Information Security Journal: A Global Perspective,* vol. 24, no. 4-6, pp. 194-206, 2015.

Appendix A: NVD Properties

Vulnerability Specification:

Property	Description
id	Unique identifier for every reported vulnerability
assigner	The assigner organization
problemtype	List of problems based on CWE
references	Report reference (e.g. HackerOne)
description	Description of vulnerability
configuration	Product configuration (e.g. versioning) which the vulnerability applies
impact	Impact score based on different metrics (e.g. CVVS3)
publishDate	Initial publish date
lastModifiedDate	Date of last modification

Table 0-1: Vulnerability Specification

CWE Specification:

Property	Description
id	Unique identifier for weakness
name	Title of weakness
description	Full description of weakness
related	Relationships between different weaknesses (e.g., ChildOf)
applicablePlatforms	Vulnerable platforms to this weakness
modesOfIntroduction	Level of vulnerability
commonConsequences	Common consequences of exploit
likelihoodOfExploit	How likelihood is to be exposed
potentialMitigation	Mitigation steps to prevent exploit

Table 0-2: CWE Specification

CPE Specification:

Property	Description
deprecated	Is product deprecated?
cpeUri	Unique identifier of exact product and release information
lastModifiedDate	Date of last modification
titles	Product title in multiple languages
refs	Product reference links
vulnerabilities	Known vulnerabilities for the product

Table 0-3: CPE Specification

Appendix B: HackerOne Properties

Bounty Program Specification:

Property	Description
url	Program page info on HackerOne website
name	Project/Company name
about	A short description of project/company
stripped_policy	Company policy
disclosure_url	Where users can report their activity
offers_reward	If the program offers any reward
offers_thanks	If the program offers any thanks

Table 0-1: Bug Bounty Specification

Bounty Report Specification:

Property	Description
reporter	The hacker who reported
state	Report current state
disclosion_date	The date of disclosion the vulnerability
weakness	Exposed weaknesses
cve	The issued cve as result of the activity
severity	Severity score
bounty	Amount of bounty paid to the hacker

Table 0-2: Bounty Report Specification

Bounty Hunter Specification:

Property	Description
name	User provided full name
hackerOneId	User id in HackerOne
joinedAt	Date of HackerOne sign up
username	HackerOne username
workplaceHomePage	User description page

Table 0-3: Bounty Hunter Specification

Appendix C: Libraries.io Properties

Library Specification:

Property	Description
name	Project name
dependentReposCount	Number of dependent repositories
deprecationReason	Is it deprecated or not and why
description	A project description
forks	Number of forks of the repo
homepage	Project homepage URL
keywords	Project keywords
language	The main programming language
latestDownloadUrl	Latest URL of built project
latestReleaseNumber	Latest release version number
latestReleasePublishedAt	Latest release date
latestStableReleaseNumber	Latest stable release version number
latestStableReleasePublishedAt	Latest stable release date
license	Project license
packageManagerUrl	Package manager URL
stars	Number of repository stars
dependencies	List of project dependencies

Table 0-1: Library Specification