

ONTOLOGY BASED SECURITY THREAT ASSESSMENT AND
MITIGATION FOR CLOUD SYSTEMS

Patrick Kamongi

Dissertation Prepared for the Degree of
DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

December 2018

APPROVED:

Krishna Kavi, Major Professor
Mahadevan Gomathisankaran, Committee
Member
Song Fu, Committee Member
Hassan Takabi, Committee Member
Barrett Bryant, Chair of the Department of
Computer Science and Engineering
Yan Huang, Interim Dean of the College of
Engineering
Victor Prybutok, Dean of the Toulouse
Graduate School

Kamongi, Patrick. *Ontology Based Security Threat Assessment and Mitigation for Cloud Systems*. Doctor of Philosophy (Computer Science and Engineering), December 2018, 155 pp., 16 tables, 80 figures, 126 numbered references.

A malicious actor often relies on security vulnerabilities of IT systems to launch a cyber attack. Most cloud services are supported by an orchestration of large and complex systems which are prone to vulnerabilities, making threat assessment very challenging. In this research, I developed formal and practical ontology-based techniques that enable automated evaluation of a cloud system's security threats. I use an architecture for threat assessment of cloud systems that leverages a dynamically generated ontology knowledge base. I created an ontology model and represented the components of a cloud system. These ontologies are designed for a set of domains that covers some cloud's aspects and information technology products' cyber threat data. The inputs to our architecture are the configurations of cloud assets and components specification (which encompass the desired assessment procedures) and the outputs are actionable threat assessment results. The focus of this work is on ways of enumerating, assessing, and mitigating emerging cyber security threats. A research toolkit system has been developed to evaluate our architecture. We expect our techniques to be leveraged by any cloud provider or consumer in closing the gap of identifying and remediating known or impending security threats facing their cloud's assets.

Copyright 2018
by
Patrick Kamongi

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisors Professor Krishna Kavi and Dr. Mahadevan Gomathisankaran for their unwavering support during my Ph.D. study and research. I am forever grateful to them for providing their guidance and sharing their wisdom with me during my Ph.D. journey. They have inspired me to become a better scholar and a well-rounded person.

Besides my advisors, I would like to thank the rest of my dissertation committee, Dr. Song Fu and Dr. Hassan Takabi, for their guidance and invaluable collaborations during my Ph.D. program.

My sincere thanks also go to Dr. Barrett Bryant, and the Computer Science and Engineering department faculty and staff members.

I thank my colleagues in the Computer Systems Research Laboratory for the good memories and experiences that we shared. Special thanks to Dr. Srujan Kotikela, Dr. Chen-Yu Lee, David Struble, and Rohith Yanambaka Venkata for their collaboration on many research projects.

Last but not the least, I would like to thank my family and friends.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	iii
LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 RELATED WORKS	5
CHAPTER 3 ONTOLOGY ENGINEERING APPROACH	12
3.1. Overview	12
3.2. Ontologies Design	13
3.2.1. Cloud Computing System – Domain	13
3.2.1.1 A Look into the HOST Class of Our Cloud System Ontology	16
3.2.2. Cyber Threat Data – Domain	18
3.3. Instantiating Ontologies	21
3.3.1. OpenStack Based Cloud System – OKB	23
3.3.1.1 Other Complex System Deployment Examples – OKBs	26
3.3.2. Cyber Threat Data – OKB	28
3.4. Tools for Ontology Engineering	37
3.4.1. IKAWAFARM	37
3.4.2. LEGOS	38
3.5. Basic Query Operations on Our Ontology Knowledge Bases	38
CHAPTER 4 ONTOLOGICAL BASED VULNERABILITY ASSESSMENT	42
4.1. Overview	42
4.2. Framework to Assess Cloud System Vulnerabilities	42
4.2.1. VULCAN – Architecture	44

4.2.1.1	IT Products' Vulnerability Data Feeds – NVD	45
4.2.1.2	Vulnerability Ontology Knowledge Bases	45
4.2.1.3	Cloud System Classifiers	46
4.2.1.4	Vulnerability Based Indexing Techniques	47
4.2.1.5	Vulnerability Index Knowledge Bases	48
4.2.1.6	Vulnerability Assessment Report Generator	48
4.2.1.7	Semantic Natural Language Processing (SNLP)	49
4.2.1.8	Other Components	49
4.2.2.	VULCAN - Implementation	50
4.2.2.1	Vulnerability Based Indexing Techniques	50
4.2.2.2	Vulnerability Assessment Report Generator	51
4.2.2.3	Semantic Natural Language Processing	52
4.2.2.4	VULCAN Web Application	53
4.2.3.	VULCAN - Evaluation	54
4.3.	Ranking Cloud System Vulnerabilities	65
4.3.1.	Overview	65
4.3.2.	Background	67
4.3.2.1	On-Fly Authentication Application	67
4.3.2.2	Amazon Elastic Compute Cloud (EC2)	67
4.3.3.	Our Approach for Security Risk Quantification	68
4.3.3.1	Attack Graph Generation	68
4.3.3.2	Common Vulnerability Scoring System (CVSS)	71
4.3.4.	Example Workflow	71
4.3.5.	Summary	74
4.4.	Prediction Model for Unknown Vulnerabilities	74
4.4.1.	Overview	75
4.4.2.	Our Predictive Methodology	76
4.4.2.1	Data Collections	76

4.4.2.2	Predictive Model	77
4.4.2.3	Prediction Workflow	79
4.4.3.	Experimental Study and Evaluation	80
4.4.3.1	Use Case: OpenSSL – Software Product	80
4.4.3.2	OpenSSL – Dataset Generation	80
4.4.3.3	OpenSSL – Predictive Experiment	81
4.4.3.4	OpenSSL – Predictive Model Validation	84
4.4.3.5	OpenSSL – Prediction of Unknown Vulnerabilities	86
4.4.3.6	Discussions	86
4.4.3.7	Recommended Proactive Strategies	88
4.4.4.	Summary	88
CHAPTER 5 ONTOLOGICAL BASED THREAT MANAGEMENT		90
5.1.	Overview	90
5.2.	Background	91
5.2.1.	Threat Modeling	91
5.2.2.	Risk Assessment	93
5.3.	Architecture	94
5.3.1.	Design	94
5.3.1.1	VULCAN Framework	96
5.3.1.2	STRIDE Threat Model	96
5.3.1.3	Bayesian Threat Probability Model	96
5.3.1.4	Nemesis’s Lightweight Applications	97
5.3.2.	Implementation	98
5.4.	Experiments and Evaluations	105
5.4.1.	Experiments	105
5.4.2.	Evaluations	106
5.4.2.1	Risk Estimator Application	106
5.4.2.2	Severity Ranking Engine Application	106

5.4.2.3 Exploitable Vulnerabilities Generator Application	107
5.4.2.4 Suggested Configurations Generator Application	107
5.4.2.5 Challenges	107
5.5. Summary	109
CHAPTER 6 COCKATOO SYSTEM	111
6.1. Overview	111
6.2. Architecture	112
6.3. Implementation	117
6.4. Evaluation	123
6.4.1. IT Assets Inventory	123
6.4.2. Vulnerability Assessment	127
6.4.3. Risk Assessment	136
6.5. Discussion and Summary	139
CHAPTER 7 CONCLUSION AND FUTURE WORK	142
REFERENCES	145

LIST OF TABLES

	Page
Table 4.1. VULCAN Web Application - OpenStack:Newton Vulnerability Indexes OKBs Comparison	57
Table 4.2. Using our Labeled OpenSSL Dataset to Evaluate its Boosted Decision Tree Regression Model	85
Table 4.3. Sample of Scored OpenSSL Instances	85
Table 4.4. Sample of OpenSSL Instances for Validation	86
Table 4.5. OpenSSL's Versions – Predicted Number of Vulnerabilities	87
Table 5.1. STRIDE Mitigation Techniques	93
Table 5.2. Our Sample Cloud Configuration	106
Table 5.3. Threat Types - Severity Rank Evaluations	107
Table 5.4. Exploitable Vulnerabilities Metric Evaluations	108
Table 5.5. Alternative Recommended Cloud Configuration	109
Table 6.1. COCKATOO System – Python Codebase Distribution	123
Table 6.2. Example of IT Systems	124
Table 6.3. LEGOS – Bitnami WordPress V0.9 Software Stack Example	126
Table 6.4. Number of Triples of Deployed IT Systems OKBs – Generated by our LEGOS Agent	127
Table 6.5. IT Systems Vulnerability Index OKBs – Generated by VULCAN	133
Table 7.1. COCKATOO Tools – Future Extensions	144

LIST OF FIGURES

	Page
Figure 3.1. Cloud Computing System Ontology	15
Figure 3.2. Generic - Host Ontology	17
Figure 3.3. Cyber Threat Data Ontology	19
Figure 3.4. Extended - Cyber Threat Data Ontology	20
Figure 3.5. STIX Ontology	21
Figure 3.6. OpenStack Deployment - Hosts Ontology Knowledge Base	24
Figure 3.7. OpenStack Deployment - Hosts Ontology Knowledge Base – Sample	25
Figure 3.8. OpenStack-Netwon Release – Cloud System Ontology Knowledge Base	26
Figure 3.9. OpenStack-Queens Release – Cloud System Ontology Knowledge Base	27
Figure 3.10. Ubuntu Server Deployment - Host Ontology Knowledge Base	28
Figure 3.11. Ubuntu Server Deployment - Host Ontology Knowledge Base – Sample	29
Figure 3.12. Lamp Stack Deployment - Host Ontology Knowledge Base	30
Figure 3.13. Lamp Stack Deployment - Host Ontology Knowledge Base – Sample	30
Figure 3.14. Kubernetes Deployment - Host Ontology Knowledge Base	31
Figure 3.15. Kubernetes Deployment - Host Ontology Knowledge Base – Sample	32
Figure 3.16. Statistic Summary of Our Studied and Deployed Systems’s - Auto Generated Host Ontology Knowledge Bases	33
Figure 3.17. Host Ontology Knowledge Base - Graphical SPARQL Query Example	33
Figure 3.18. Cyber Threat Data Ontology Knowledge Base – Sample	34
Figure 3.19. Cyber Threat Data Ontology Knowledge Base	34
Figure 3.20. McAfee TLD Ontology – High Level View	35
Figure 3.21. McAfee TLD Ontology Knowledge Base – Vulnerabilities Instances View	35
Figure 3.22. McAfee TLD Ontology Knowledge Base – Sample View	36
Figure 3.23. McAfee TLD Ontology Knowledge Base – Extended View	36
Figure 3.24. Sample of Statements with HasCveSummary as a Predicate	39
Figure 3.25. Indexed Graph – OpenStack Queens Release	41

Figure 4.1.	VULCAN Architecture	44
Figure 4.2.	VULCAN – Semantic Natural Language Processor (SNLP)	49
Figure 4.3.	VULCAN Web Application - OpenStack:Newton Legos Generated OKB Input	57
Figure 4.4.	VULCAN Web Application - OpenStack:Newton Generated Vulnerability Indexes OKBs	58
Figure 4.5.	VULCAN Web Application - OpenStack:Newton Vulnerability Assessment Report using a Generated Vulnerability Index Level 3 OKB	58
Figure 4.6.	VULCAN Web Application - OpenStack:Newton Executive Summary View	59
Figure 4.7.	VULCAN Web Application - OpenStack:Newton Assets View	59
Figure 4.8.	VULCAN Web Application - OpenStack:Newton Vulnerable Installed Packages per Host	60
Figure 4.9.	VULCAN Web Application - OpenStack:Newton Found Vulnerability for a Selected Installed Package	60
Figure 4.10.	VULCAN Web Application - OpenStack:Newton Selected Vulnerability Identifier Details	61
Figure 4.11.	VULCAN Web Application - OpenStack:Newton Vulnerability View – Sample	61
Figure 4.12.	VULCAN Web Application - OpenStack:Newton Selected Vulnerability Identifier Details	62
Figure 4.13.	VULCAN Web Application - OpenStack:Newton Exploit View	62
Figure 4.14.	VULCAN Web Application - OpenStack:Newton Selected Exploit Identifier Details	63
Figure 4.15.	VULCAN Web Application - OpenStack:Newton Selected Vulnerability Identifier Details	63
Figure 4.16.	VULCAN Web Application - OpenStack:Newton Chatbot First User Interaction	64

Figure 4.17.	VULCAN Web Application - OpenStack:Newton Chatbot Sample Answer to a User Query	64
Figure 4.18.	VULCAN Web Application - OpenStack:Newton Chatbot Sample Answer to a User Query	64
Figure 4.19.	Vulnerabilities Ranking Methodology	66
Figure 4.20.	On-the-Fly Authentication Application – User Interface	68
Figure 4.21.	On-the-Fly Authentication Application – Example	69
Figure 4.22.	On-the-Fly Authentication Application – Example Evaluation	69
Figure 4.23.	Predictive Model - Framework	78
Figure 4.24.	OpenSSL Releases vs. Disclosed Number of Vulnerabilities	82
Figure 4.25.	OpenSSL:1.0.1 – Minor Releases Linear Trend	83
Figure 5.1.	NEMESIS - Architecture	95
Figure 5.2.	Utilizing OKBs for the Bayesian Threat Probability Determination	98
Figure 5.3.	Equations for an Ontology Based and Bayesian Threat Probability Determination – Approach	99
Figure 6.1.	COCKATOO Tools Dependency Graph	113
Figure 6.2.	COCKATOO System Workflow	114
Figure 6.3.	COCKATOO System – Web Portal	117
Figure 6.4.	COCKATOO System – Technology Stack Sample	118
Figure 6.5.	COCKATOO Tools – Gallery Preview	118
Figure 6.6.	COCKATOO Tools – Use Cases	124
Figure 6.7.	COCKATOO – LEGOS Tool	125
Figure 6.8.	LEGOS Agent – Datastore View of Auto-generated IT Systems OKBs	126
Figure 6.9.	COCKATOO – VULCAN Tool	128
Figure 6.10.	COCKATOO – HUMMING Tool	128
Figure 6.11.	VULCAN – Portal	129
Figure 6.12.	VULCAN – OpenStack Vulnerability Assessment Report	130
Figure 6.13.	OpenStack Vulnerability Assessment Report’s Executive Summary	130

Figure 6.14.	OpenStack Vulnerability Assessment Report's Asset View	131
Figure 6.15.	OpenStack Vulnerability Assessment Report's Vulnerability View	131
Figure 6.16.	OpenStack Vulnerability Assessment Report's Exploit View	132
Figure 6.17.	OpenStack Vulnerability Assessment Report's Mitigation View	132
Figure 6.18.	OpenStack Vulnerability Assessment Report's Visualization View	133
Figure 6.19.	HUMMING – Dialogue Example	134
Figure 6.20.	COCKATOO – SWEEP Tool	134
Figure 6.21.	COCKATOO – PREDICTION Tool	135
Figure 6.22.	OpenSSL – Predictive Experiment Example	135
Figure 6.23.	COCKATOO – NEMESIS Tool	136
Figure 6.24.	NEMESIS Portal with a User Input of their IT System Vulnerability	
	Index OKB	137
Figure 6.25.	NEMESIS Threat Modeling – Key Findings for a WordPress Stack	137
Figure 6.26.	NEMESIS Risk Assessment – Key Findings for a WordPress Stack	138
Figure 6.27.	NEMESIS Evaluation – Key Findings for an OpenStack Deployment	139

CHAPTER 1

INTRODUCTION

The cloud computing [72] model has enabled a wide range of technologies and services that have a significant impact on our society. The cloud delivery model using the internet exposes technologies and services to a large attack surface [117, 80, 15, 51, 70] which inevitably is exploited by malicious actors motivated to mount various campaigns on the cloud ecosystem and its adopters.

A report by the Cloud Security Alliance (CSA) organization [24] titled “The Treacherous 12 - Cloud Computing Top Threats in 2016” [49], lists: data breaches; weak identity, credential and access management; insecure application program interfaces (APIs); system and application vulnerabilities; account hijacking; malicious insiders; advanced persistent threats (APTs); data loss; insufficient due diligence; abuse and nefarious use of cloud services; denial of service; and shared technology issues as the most significant security issues in the cloud. This report among others [47, 48] by CSA have these same threats to cloud security in common. The “System and Application Vulnerabilities; and Shared Technology” threats, in particular, have a significant implication for cloud systems which are used to orchestrate various cloud services that are in turn consumed by users from various entities.

There are many communities and industry and research initiatives that are addressing current security issues in information technology (IT) products (systems, applications) with public disclosure of security information and data feeds (common vulnerabilities, weaknesses, and configurations [76, 77, 82]). These approaches have been adopted by some security vendors and organization teams aiming to mitigate some of the threats arising from the exploitation of weaknesses in IT products. Most of these security solutions have not been successful enough to mitigate and protect cloud assets, primarily because of the scale and complexity of the cloud computing model (its architecture, deployment, and operational choices).

In this work, we propose a viable solution to an open security problem of threats

to cloud computing. We use ontologies as the foundation for our techniques to enumerate, assess, and mitigate the threats to cloud systems. One of the core features of ontologies is the powerful modeling capability that enables representation of the knowledge body of cloud computing systems and cyber threat data domains. Ontologies are computational artifacts which are machine readable and can be associated with formal meanings. By using a semantic query language (e.g., SPARQL [101]) and a reasoner (e.g., HermiT [43]), we can automatically ascertain some facts about cloud systems and their security postures. This, in turn, allows us to build several frameworks for assessing the threats facing any given cloud system.

Our work thus far has resulted in these original key peer-reviewed papers:

- VULCAN: Vulnerability Assessment Framework for Cloud Computing [92].
- A Methodology for Ranking Cloud System Vulnerabilities [93].
- NEMESIS: Automated Architecture for Threat Modeling and Risk Assessment for Cloud Computing [91].
- Predicting Unknown Vulnerabilities using Software Metrics and Maturity Models [90].
- OPTIMUS: A Framework of Vulnerabilities, Attacks, Defenses, and SLA Ontologies [17].

Our research also resulted in the following prototype software packages:

- *IKAWAFARM*: a tool for designing scalable ontologies and their semantic knowledge graphs (including their semantic query APIs) for cybersecurity threat data and cloud computing domains (See Chapter 3).
- *LEGOS*: a tool to collect various telemetry data of a computer host (i.e., physical/virtual/container machine) and store them into a designated semantic graph database (i.e., AllegroGraph [59]) instances (See Chapter 3).
- *VULCAN*: a tool for our VULCAN [92] contribution work which provides an on-demand vulnerability assessment web application (See Chapter 4).
- *HUMMING*: a semantic natural language processing Chatbot that augments Vulcan-

Application to provide vulnerability information to users (See Chapter 4).

- *NEMESIS*: a tool for our NEMESIS [91] contribution work offered as a web application (See Chapter 5).
- *SWEEP*: a tool to automate the process of software complexity metrics generation and analysis for any given IT software product along with its vulnerability history timeline, for building a machine learning model to predict the number of unknown vulnerabilities associated with the product (See Chapter 4).
- *PREDICTION*: a tool that offers a web service to predict the unknown number of vulnerabilities in a software product (See Chapter 4).
- *COCKATOO*: an integrated toolchain offered as a service to assess and mitigate threats facing any computing system (See Chapter 6). This tool creates a workflow based on user needs and invokes previously mentioned tools.

We are able to manage a cloud system’s complexity using our ontology design approach, which is based on our understanding of cloud computing in terms of its model definition, architecture, and deployment. A similar approach to modeling the cyber threat data domain enables us to overlay the cloud system’s ontology and assess its vulnerabilities systematically. Our techniques enable us to model any given cloud system deployment in terms of known or unknown vulnerabilities and threat types enabled by these vulnerabilities. This enables us to estimate the overall cloud system’s risk posture using both qualitative and quantitative approaches. By leveraging cyber threat intelligence, the estimated risk can be computed and used in managing mitigation strategies based on cost and severity of specific threats. Our tools are used to provide recommendations to mitigate specific threat types proactively.

The rest of the dissertation is presented as follows. In Chapter 2, we present the most salient related works pertinent to our study. Chapter 3 presents our ontologies for describing IT systems and for capturing cyber security vulnerabilities. In Chapters 4 and 5, we present our approach for vulnerability assessment and threat management. Chapter 6 presents our custom built COCKATOO system, which is a toolchain that integrates all our contributions.

We present our concluding remarks and future work in Chapter 7.

CHAPTER 2

RELATED WORKS

In this chapter, we present key related works that leverage ontological foundations to address important cybersecurity problems.

Miguel-Angel et al. [114] have conducted a literature survey to highlight some of the recent works in regards to what extent ontologies are useful for addressing various information security problems. In this work, the authors organized the surveyed papers into these categories: security requirement engineering, reference ontologies, specification and matching of security policies and access control, vulnerability analysis, attack detection, information extraction, and preparation for machine learning. This study aimed to identify different ways security-themed ontologies have been proposed and demonstrated as useful tools, and to open dialogue towards the value and role that ontologies offer in the information security field. Then, the authors point out that ontology population and the value of using ontology-based reasoning capabilities over existing formalisms or databases are some of the main issues that need to be addressed and studied further to strengthen the application and maturity of security ontologies. In chapter 3, we present our contribution work in details that address some of the challenges of ontology population and reasoning at scale.

Syed et al. [116] created a Unified Cybersecurity Ontology (UCO) to support information integration from a variety of cybersecurity standards, and to support cyber situational awareness in cybersecurity systems. UCO ontology design principles share similarities with our created ontology artifacts (See Chapter 3) for the cyber and cloud domains in terms of the choice of ontology languages (RDF [102], and OWL [100]) and data-feed sources (e.g., NVD [85], STIX [88], etc.) to generate a rich semantic knowledge base to support various use cases. Similar to the authors' UCO ontology use cases that revolve around vulnerability information associated with IT products and their vendors, we developed automated ontology-based tools (See Chapter 6) to address these use cases and more in detail by abstracting from a user the underlying semantic queries and reasoning to answer any question

that a given user may have in regards to their IT system’s security posture. Also, our original VULCAN [92] ontologies predate UCO.

Iannacone et al. [53] developed an ontology (called “STUCCO”) for a cybersecurity knowledge graph database. In this work, the authors presented how their ontology addresses a number of challenges that current security professionals face vis-à-vis the large volume of threat data (including their diverse data-feed sources, and lack of interoperability in between these sources). This work shares with our contribution, the same philosophy of designing a custom ontology to represent a domain of interest (cybersecurity) with a use case in mind, and generating a knowledge graph database to provide ground truths to automated systems that address a number of security problems that are hard to perform manually.

Takahashi et al. [118] proposed an ontological approach to cybersecurity in cloud computing to address the need for security best practices within the cloud domain. They created an ontology for cybersecurity operational information that enables understanding of security challenges that need to be addressed within the cloud domain. We have proposed novel ontology-based solutions to address some of the cybersecurity needs listed in this work [118] such as cloud resources mapping to an underlying technology stack and the interdependencies between cloud services, and acquisition of cyber risk, countermeasures, and assets knowledge bases. For instance, in Chapter 3 and Chapter 6, we present our ontology models for the cloud and cyber domains, and tools built to generate cyber threat data and cloud system ontology knowledge bases to manage and assess vulnerability-based threats facing any given organization cloud computing IT assets.

Steele’s [115] work on ontological vulnerability assessment shows that taking an ontological approach results in improved identification of complex vulnerabilities. In our work, we are able to query and reason on our ontology knowledge bases (generated by our IKAWA-FARM tool, See Section 3.4.1) to find known vulnerabilities and discover unknown ones for a given target system.

Guo et al. [52] present an ontology-based approach to model security vulnerabilities listed in Common Vulnerabilities and Exposures (CVE), providing machine-understandable

CVE vulnerability knowledge and reusable security vulnerabilities interoperability. Their efforts to form a well-structured ontology which includes concepts, concept taxonomies, relationships, properties, axioms, and constraints allowed us to extend their work into our ontology models' definition.

The Ontology for Vulnerability Management (OVM) [120] captures important concepts and relations for describing vulnerabilities in the context of software and system security. Their implementation of an ontology in Web Ontology Language - Description Language (OWL-DL) uses Protege [37]. This task can become time-consuming when looking at manually instantiating the ontology from a big data source like NVD [85]. To overcome this challenge, we generated our cyber threat data ontology knowledge bases (OKBs) using custom Python scripts to extract relevant data from the NVD data feed and populate our ontology model automatically (See Section 3.3). Our generated OKBs allow us to find and assess vulnerabilities in cloud systems.

Wang et al. [121] proposed an ontology-based approach to analyze and assess the security posture for software products. Given a knowledge base of security vulnerability information, a user can query and retrieve currently known vulnerabilities of a given target software product. In our work, we are able to find known vulnerabilities in a given software product by querying our generated cyber threat data ontology knowledge base. Also, we leverage ontology-based reasoning techniques to discover new vulnerabilities for the given software product through inferences from a generated cyber threat data OKB, and a user's deployed IT system's knowledge base.

The state-of-the-art automatic ontology generation [14] defines its life cycle as a process composed of Extraction (acquire information needed to generate the ontology), Analysis (focus on the matching of retrieved information and/or alignment of two or more existing ontologies, depending on the use case), Generation (generate the Ontology), Validation (authenticate whether the generated ontology is correct or not), and Evolution (adapt to the ontology changes). In our work, we designed and implemented an algorithm that allows us to automate ontology knowledge base generation and population from multiple sources of

vulnerability data-feeds (See Section 3.3) and checked for their consistency using the HerMiT reasoner [43]. In addition, we designed a toolkit to maintain our ontology knowledge bases (See Section 3.4.1).

Meunier’s [73] work presents a survey of currently known attempts to classify vulnerabilities and attacks. It illustrates how the traditional classifications fail to come up with one unified classification schema of all vulnerabilities and attacks. A recommended solution to this problem is to use an ontology for vulnerabilities conceptualization. A well-defined ontology model is capable of representing all kinds of vulnerabilities regardless of which subcategories they belong to. Our custom ontology model proves that the recommended approach is useful when developing a vulnerability analysis assessment framework (See Section 4.2).

Attack graphs depict ways in which an adversary exploits system vulnerabilities to achieve a desired state [108]. Sheyner et al. proposed a tool useful for generating and analyzing attack graphs. In our work, we rely on our generated ontology knowledge bases as sources for the ground truth to discover known vulnerabilities that affect a given target system, then initialize the attack graph generation for it (See Section 4.3).

The metrics we use (See Section 4.3) to model a given IT system’s found vulnerabilities are based on the Common Vulnerability Scoring System (CVSS) [36] and Weighted mean [123]. The combination of CVSS and weighted mean metrics allow us to generate a collective metric system for multiple vulnerabilities that build on the discovered attack paths. In Wang et al. [122] work on attack graph-based probabilistic security metrics assigns probabilities to each attack path’s nodes and using their defined algorithm they are able to compute the cumulative probabilities to reach the attack end goal. However, finding probabilities for each attack path node is a challenge [109] since each vulnerability associated with a node is an individual entity and having it correlate with others is not applicable. To address this challenge, we introduced weighted mean to the given attack path so that a user can provide some additional information that will give more context to our ranking algorithm 2; that will compute the resulting attack paths efficiently while conserving each

attack node’s metrics.

Yonghee et al. [113] attempted to understand whether there is a correlation between software complexity measure and security, primarily focusing on the JavaScript Engine in the Mozilla application framework. They show a weak correlation, primarily because of the small number of features used. In our study (See Section 4.4), we expanded on the number of software metrics and used product releases to obtain higher correlations to reported vulnerabilities. Our unknown vulnerability predictive model (See Section 4.4.2.2) works well when there is a large number of product releases, and the product has a mature user base.

Other prior works have explored various software properties to serve as indicators of vulnerabilities where they used techniques such as software complexity and developer activity metrics [111][110][18]. Then using these software metrics coupled with some empirical models, there are works [112][6][4][5] that have proposed solutions towards representing and predicting trends in software vulnerabilities. Our research work (See Section 4.4) has some key concepts similar to these prior works, but we extend the scope regarding automatic data analysis and vulnerability prediction.

Donevski et al. [28] work on analyzing virtual machine security threats that might arise due to a multi-tenant architecture with a given cloud environment. The proposed approach revolved around discovering new vulnerabilities for different networking configurations within the cloud network controller using a third-party security assessing tool. Though this work’s approach to performing threat modeling is limited, it points out a critical threat variant of new vulnerabilities that arise from the composition of shared technologies within a cloud setting. In our proposed NEMESIS architecture (See Chapter 5), we show how these new types of vulnerabilities come into the picture, and how we perform a detailed threat modeling for each found vulnerability, and how all found vulnerabilities impact the threat level of any given cloud setting.

Amartya et al. [106] work on “Off-line Risk Assessment of Cloud Service Provider” have in common with our proposed architecture (See Chapter 5) in terms of their methodology of assessing threats present in a client’s application/cloud service using the STRIDE

model [62]. Our approach automates the threat modeling methodology of using the EoP card game [23] based STRIDE model, instead of manually generating threat types as has traditionally been done as in the Amartya et al. [106] approach. In addition, our risk assessment approach within the NEMESIS architecture is automated.

In Fenz [34], [35] and Settas et al. [107] works have two main components in common with our proposed NEMESIS architecture (See Chapter 5) such as the use of an ontological approach and Bayesian network key building blocks towards various security assessment works. Additional ontological security related works by Fenz et al. that motivated us to build our NEMESIS's risk assessment model are:

- Ontology-based generation of IT-security metrics [33]; here we use various measurements generated from NEMESIS's Vulcan and Threat modeling subroutines to compute and produce useful metrics regarding the status of a given cloud's assets such as: "Threat type's severity average score", and "Unpatched critical vulnerabilities" metrics for any given cloud system.
- Ontology and Bayesian-based threat probability determination [119], which we modified for the cloud setting and automated it into our NEMESIS's threat ranking subroutine; and
- Security ontology: Simulating threats to corporate assets [30]; here we perform a thorough threat modeling task and generate all possible threat types for all discovered vulnerabilities for any given cloud system.

Farris et al. [32] work on VULCON presents a system for vulnerability prioritization, mitigation, and management. This work focuses on designing a vulnerability scoring system that takes into account the context information about the nature of the vulnerability, criticality of the impacted service, and the workforce available within a given organization to patch the vulnerabilities in a timely manner. This scoring system is useful to prioritize vulnerabilities and minimize vulnerability exposure of the IT organization's systems. In our work, we took a different approach that builds a scoring system (implemented within our COCK-ATOO system, See Chapter 6) to prioritize vulnerabilities that affect a given organization's

IT system based on the threat probability of exploitation of any found vulnerability-based threat instance. Also, based on a user's need, we can implement the VULCON scoring system in our COCKATOO solution (since many building blocks of VULCON system are built and offered within our COCKATOO system).

Noel et al. [87] work on CyGraph present a graph-based analytics and visualization solution for cybersecurity. The CyGraph approach to prioritizing a given organization's IT system's exposed vulnerabilities is similar to our work on COCKATOO's NEMESIS tool (See Figure 6.23). We prioritize found vulnerabilities that affect the given organization's IT assets (e.g., cloud system) by threat modeling all vulnerability-based threat type's instances (using a STRIDE model [62]), and for each modeled threat type instance, assess its threat probability of exploitation using an ontology and Bayesian model. Then, a user can explore NEMESIS tool recommendations on how to mitigate each threat instance. The recommendations revolve around suggesting to the user alternate IT system configurations that will minimize any perceived risk, and also a prioritized list of vulnerabilities to patch (which will close the exposed threat type instance). Also, our COCKATOO system (See Chapter 6) acts as an analytic platform powered by ontology knowledge bases. At the moment, contextual graphs generated by the COCKATOO tools can be visualized using offline tools such as Protege [37] and Gruff [60]. We can also use an approach similar to CyGraph to visualize important findings on-demand within our COCKATOO dashboard.

CHAPTER 3

ONTOLOGY ENGINEERING APPROACH

3.1. Overview

An ontology is a formal, explicit specification of a shared conceptualization [50]. Ontology specifies a set of representation primitives that allow us to model a domain of knowledge. By selecting a suitable ontology language, we can explicitly represent any given field by designing one or multiple ontologies to describe the intended application use. These ontologies should capture the concepts, attributes, and relationships that are presumed to exist in the modeled domain, along with their formal semantics. These ontologies can then be instantiated to generate knowledge bases of the modeled system of interest. Since the generated ontology knowledge bases (OKBs) are machine-readable, there are applications (e.g., querying services, intelligent agents, etc.) that can leverage them. Also, the created ontologies and generated OKBs can be reused and extended for other uses.

Ontological engineering refers to the “set of activities that concern the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them” [45]. In this chapter, we present the ontology engineering approach we used to design, create and instantiate various ontologies of interest that serve as the pillars of this dissertation. We utilize:

- Web Ontology Language (OWL 2) [100], and Resource Description Framework (RDF 1.1, & RDFS 1.1) [102, 103] as our ontology languages.
- Protege [81] as our ontology editor.
- SPARQL [101] as our query language.
- AllegroGraph [1] as our semantic graph database to create and instantiate our ontologies. The database can be used to query the ontology and reason about the generated ontology knowledge bases (OKBs).
- Gruff [60] as our graph-based Triple-Store browser for AllegroGraph.

We engineer our ontologies to model cyber threat data and cloud computing systems.

For the cyber threat data, we design our ontologies to represent knowledge about various areas that touch this domain such as vulnerabilities, exploits, mitigation, and cyber threat intelligence information. As for the cloud computing system domain, we explore the cloud computing system’s assets where we design our ontologies to represent the knowledge about all participating IT assets’ configuration. We combine the ontologies from both of our domains (cyber threat data and cloud computing systems) of interest to produce dynamically complex representations of information. These ontologies enable us to generate ontology knowledge bases that can be leveraged for performing various tasks (e.g., querying, and reasoning) to understand cyber threats and approaches to mitigate them.

3.2. Ontologies Design

We take a modular approach to create our ontologies for both cloud computing system and cyber threat data domains. The objective is to build self-contained ontologies that enable reuse and connections across our modeled domains. We start our modeling process with what we know about our domains and maintain openness for future extensions that will incorporate new knowledge and richer semantics during the ontology’s lifecycle.

3.2.1. Cloud Computing System – Domain

Cloud computing is a “model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” [72]. A cloud can be orchestrated in three service models, namely, Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) and be deployed as a Private, Public, Community, or Hybrid cloud. Any cloud deployment system is supported by a variety of technologies and a set of guidelines to govern and maintain it.

In this study, we pay particular attention to the cloud computing system fabric for a cloud provider who is designing, building and maintaining the cloud system to a consumer of the offered services. In this regard, our ontology design revolves around representing

how different information technology (IT) products are used to support cloud deployment and service orchestration. We design and create a “cloud system” ontology to capture the knowledge about IT products, and how these IT products are utilized in the deployment and orchestration.

As shown in Fig. 3.1, our cloud system ontology relies on the standard cloud computing model definition and its supporting technology specifications [72, 95]. At a high level, we want to represent the knowledge about cloud computing in terms of its participating cloud actors (provider, consumer, auditor, carrier and broker), cloud deployment models (public, private, hybrid, and community), cloud orchestration models (IaaS, PaaS, and SaaS), consumer cloud applications, and the core cloud fabric (its supporting cloud system) as represented by their respective classes (concepts) shown in the Fig. 3.1. Our IKAWAFARM tool implementation (introduced in Section 3.4.1) contains a complete ontology definition of our cloud system ontology.

The **Cpe** parent class as shown in Fig. 3.1 is based on the Common Platform Enumeration (CPE) standard and its IT product dictionary. CPE is a structured naming scheme for information technology systems, software, and packages [83]. We designed this ontology to model the existing knowledge found in the CPE standard (including naming scheme, IT product dictionary, etc.), which allows us to represent any IT product type of interest (“a: application”, “o: operating system”, and “h: hardware”) of interest that play a vital role in a given cloud ecosystem or traditional IT infrastructure.

For instance, let us consider the Ubuntu 14.04 operating system which is an example of an IT product (Ubuntu server distribution plays a critical role in most popular cloud deployments and services orchestration). Our **Cpe** class (including its sub-classes) allows us to represent what we know about Ubuntu 14.04 by utilizing the existing CPE dictionary as follows:

- We can instantiate the **CpeItem** class with one or more attributes (e.g., “cpe:2.3:o:canonical:ubuntu_linux:14.04.*:*:*:lts:*:*”))
- The above **CpeItem** class instance has other attributes that extend it using a CPE

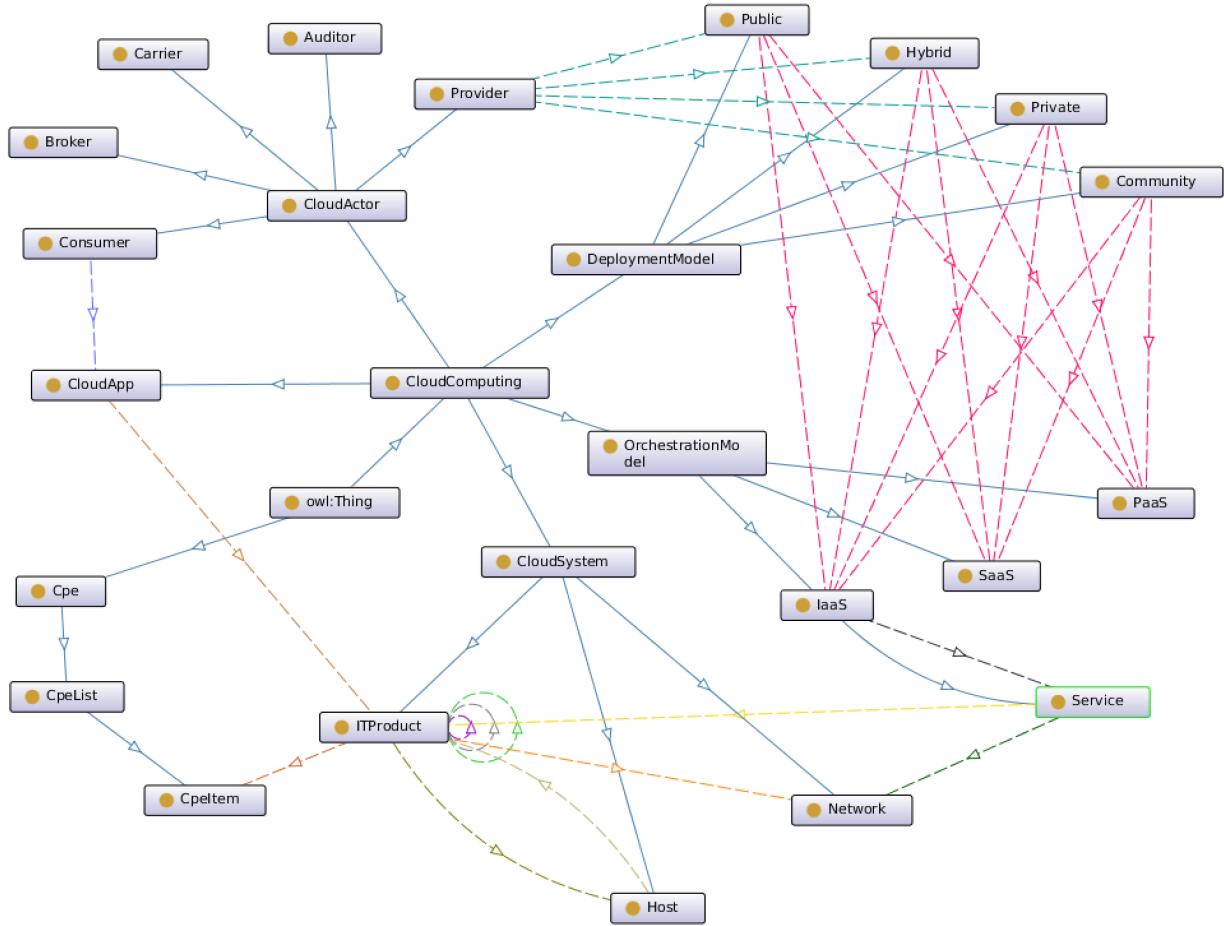


FIGURE 3.1. Cloud Computing System Ontology

well-formed name (WFN) specification (e.g., “**vendor:** canonical ”, “**product:** Ubuntu.Linux ”, “**version:** 14.04 ”, etc.)

Our ontologies enable us to instantiate our Cpe class or any other one with the information we currently have (which is sufficient to make correct assertions), and can always be updated with additional information in the future (in our above example, we can later specify any other attribute to represent an Ubuntu 14.04 Cpe instance, for example, with the details about the target hardware). The Cpe class is designed to facilitate populating class instances automatically (as discussed in Section 3.3) or instantiated manually for IT products with no CPE dictionary entries [83]).

Another key class “**CloudSystem**” of our ontology focuses on the cloud system fabric. This class acts as the parent class to three key concepts (**ITProduct**, **Host**, and

Network) to represent an active cloud system based on a specific deployment model, orchestrated services and running cloud applications. For instance, the **ITProduct** class models all individual technologies (application, hardware, and OS) and their inter-dependencies that are part of any given cloud system. Each ITProduct’ instance is linked (directly or via inferences) to other key classes such as:

- **CpeItem** class to represent the given IT product using CPE format, which enables us to infer additional information such as related IT products based on the shared CPE attributes including IT product’s vendor, version, and target hardware. This can be linked with the **Cyber Threat Data** ontology described in section 3.2.2.
- **Host** class representing information about the where the ITProduct instance is installed.
- **Network** class to represents which Network the given ITProduct instance resides in.
- **Service** class to represent the orchestrated Service instance which depends on the given ITProduct/Host/Network instance(s), and
- **CloudApp** class to represent the set of deployed CloudApp instances (each with the ITProduct class instances on which it depends on).

3.2.1.1. A Look into the HOST Class of Our Cloud System Ontology

In this section, we take a look into the **Host** class shown in Fig. 3.1 to represent the information about a Host instance (i.e., physical/virtual/container machine) that plays a key role in a cloud deployment. We designed a minimal ontology to represent information about one such generic host instance executing Ubuntu/Debian OS as an example to support various cloud system deployments (i.e., “Linux Server[69]”, “Lamp Stack[20]”, “Kubernetes[12]”, and “OpenStack[94]”).

Our Host ontology is shown in Fig. 3.2 and integrates seamlessly with the cloud computing system ontology seamlessly. The key information that we represent in this ontology includes information about the host such as:

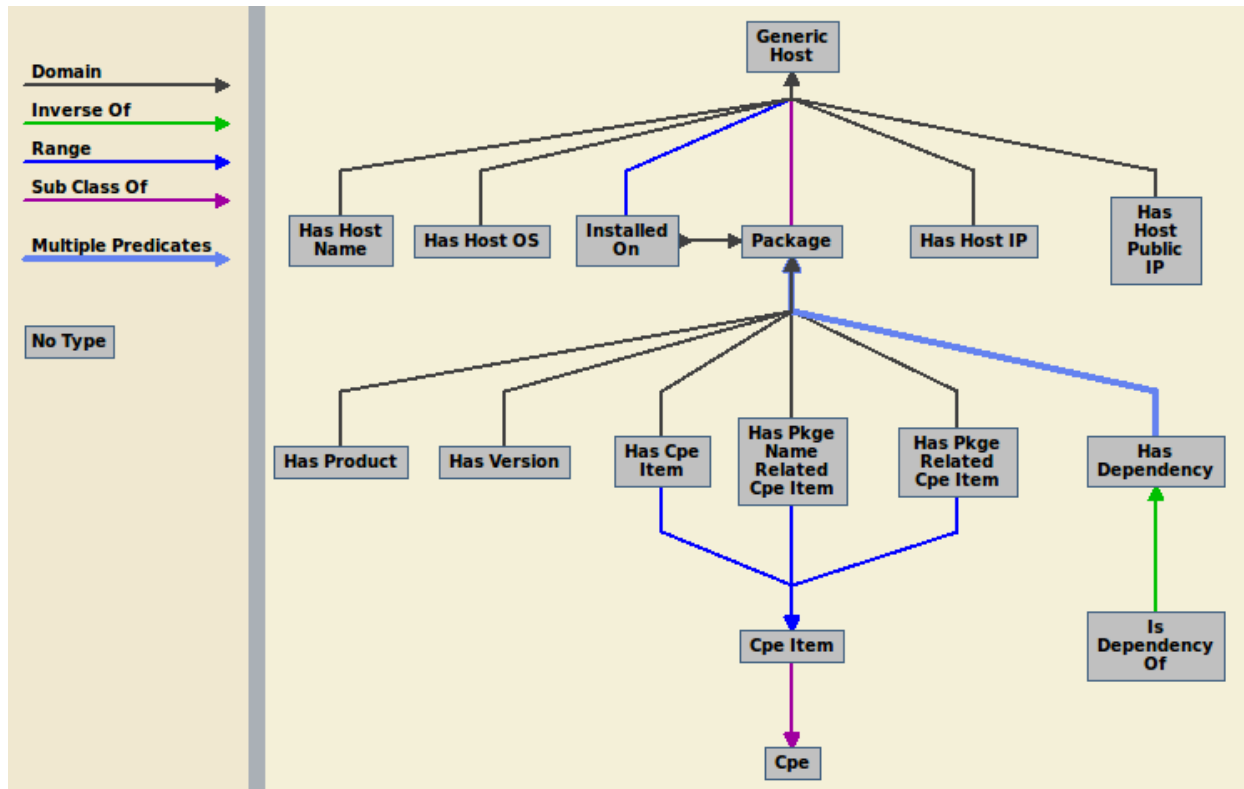


FIGURE 3.2. Generic - Host Ontology

- The hostname, executing operating system, and IPs of each host in the deployment cluster.
- All software packages installed on each host and the details of their dependencies. We define three relationship predicates *hasCpeItem*, *hasPkgeNameRelatedCpeItem*, and *hasPkgeRelatedCpeItem*:
 - *hasCpeItem* links a given package installed on the considered host to its CPE instance allowing us to discover vulnerabilities associated with this item
 - *hasPkgeNameRelatedCpeItem* enable us to link a package to all CPE instances that share the same Product name.
 - *hasPkgeRelatedCpeItem* enable us to link a package to all CPE instances that share the same Product name but Versions that are higher or equal to the current installed package’s version.

CPE instance predicates (i.e., Vendor, Product, and Version) are illustrated in our

cyber threat data ontology shown in Fig. 3.4. The three predicates play a key role in our threat assessment and mitigation techniques which are described in Chapters 4 and 5, since they allow us to link the cyber threat ontology with the cloud system ontology.

We should note that our cloud computing system ontology can be extended to represent other aspects of the cloud model. In Section 3.3, we present our approach towards populating our cloud system domain-specific ontology along with some use cases using an OpenStack [95] private cloud deployment.

3.2.2. Cyber Threat Data – Domain

IT products have been shown to be vulnerability prone [85]. As described in the previous Section 3.2.1, IT products are used to orchestrate complex IT systems such as cloud model deployments. Assessing the vulnerability of a cloud system becomes challenging because of the numerous products involved in the system and product dependencies. In this section, we present our cyber threat data ontology for capturing vulnerability information about IT products so that this information can be linked with the IT system ontology already presented. Our ontology can be populated using several publicly available databases including:

- National Vulnerability Database (NVD) [85] data-feeds
- Offensive Security Exploit Database Archive (Exploit-DB) [104] database
- Structured Threat Information eXpression (STIX) [88] data-feeds

Cyber threat data ontology is shown in Fig. 3.3 and consists of the following classes at the highest level of abstraction.

- **Nvd** to represent IT products' known vulnerability details across many aspects, where each IT product is represented in a CPE format as captured by our **Cpe** class which also exists in our **cloud system** ontology. Thus the two ontologies can be linked.
- **ExploitDB** to represent known IT products' vulnerabilities proof of concept exploits details.

- **VendorStatement** to represent published security patch information for known vulnerabilities captured by the **Nvd** class).

Our ontology can be extended to support other relevant structured or unstructured data, for example: Malware Attribute Enumeration and Characterization (MAEC) [79], Common Vulnerability Reporting Framework (CVRF) [54], and Common Attack Pattern Enumeration and Classification (CAPEC) [75].

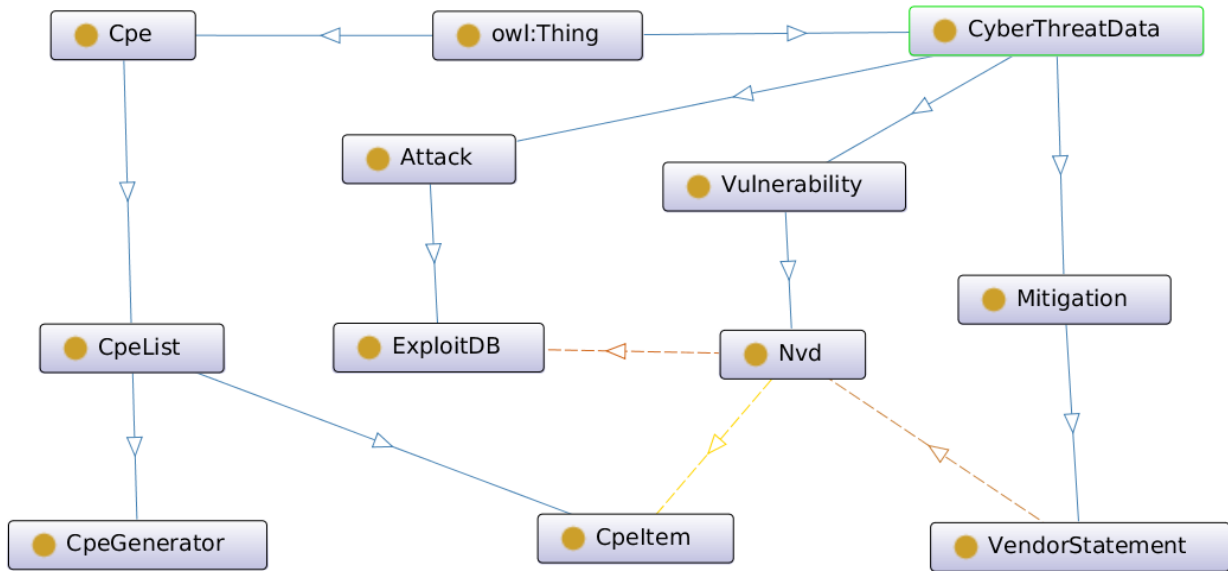


FIGURE 3.3. Cyber Threat Data Ontology

Our cyber threat data ontology is explicitly designed to model structured data sources that are embodied within the super classes (**Vulnerability**, **Attack**, and **Mitigation**). The ontology can be populated automatically with latest information available from a variety of databases as stated previously [85, 104]. We can also model unstructured cyber threat information using the STIX language [88].

STIX ontology is shown in Fig. 3.5. As of now, only a few classes (**ExploitTarget**, **Observable**, and **TTP**) have support for standard expression languages and their relevant cyber threat data sources (Common Attack Pattern Enumeration and Classification(CAPEC) [75], Malware Attribute Enumeration and Characterization (MAEC) [79], Cyber Observable eXpression (CybOX) [78], Common Vulnerabilities and Exposures (CVE)

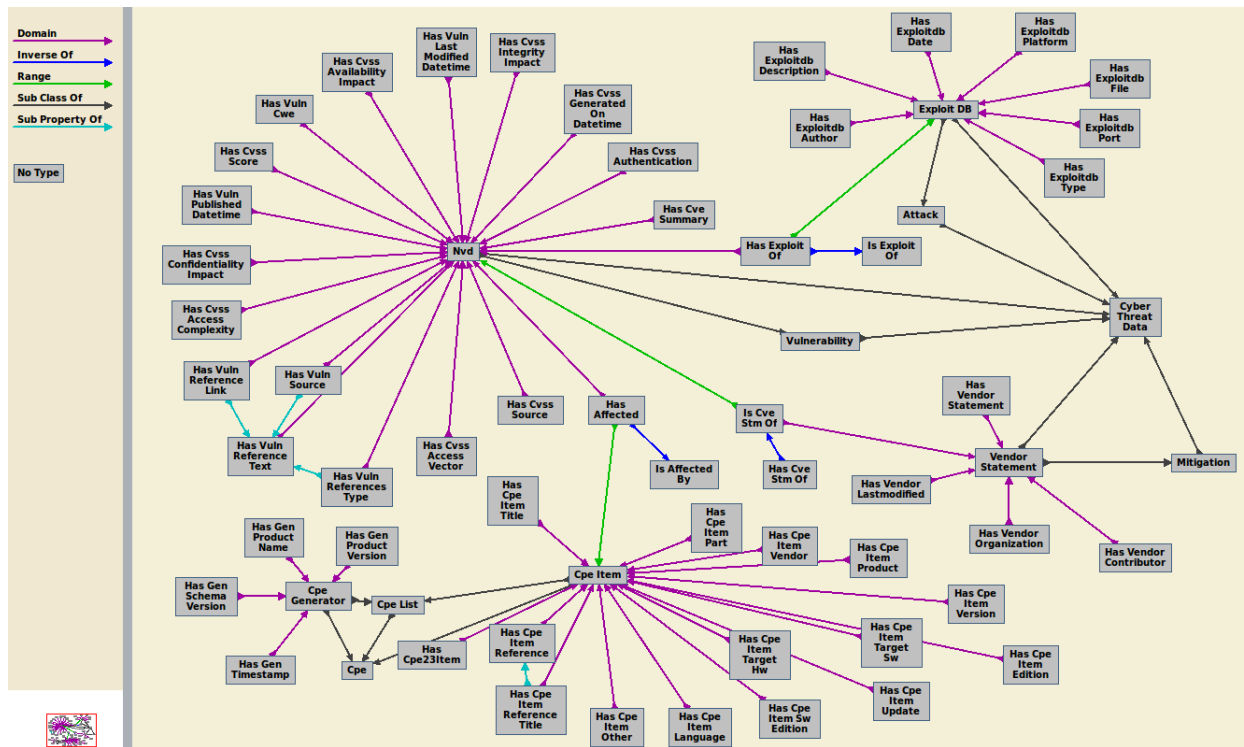


FIGURE 3.4. Extended - Cyber Threat Data Ontology

[76], Common Vulnerability Reporting Framework (CVRF) [54], Common Weakness Enumeration (CWE) [77], and Common Configuration Enumeration (CCE) [82]) while the rest rely on the users and community best practices to instantiate them. This ontology is designed to allow anyone interested in collecting and delivering raw or vetted cyber threat information to meet specific threat intelligence collection requirement while making use of the STIX language.

Within the STIX ontology, we only illustrate the high-level concepts (or classes) specified within the STIX language and their interactions. This representation approach allows anyone to define each concept/class (defining the data properties and logical constraints for each of the STIX class) using their preferred best practices or supported standard cyber threat expression languages or vocabularies.

Note that all of our instantiated ontologies along with their respective annotations are described in detail within our COCKATOO toolchain (See Chapter 6).

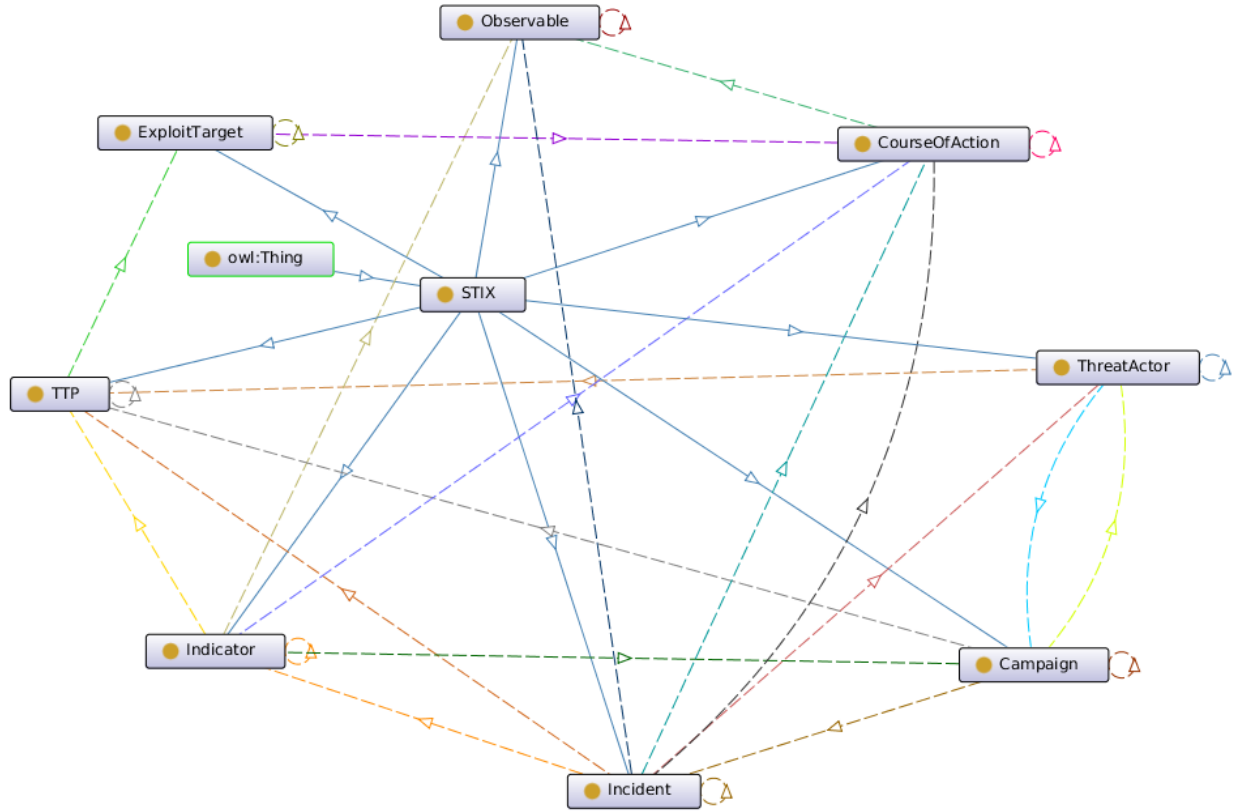


FIGURE 3.5. STIX Ontology

3.3. Instantiating Ontologies

In this section, we present our approach to populating specific instances of our ontologies (See. Figures 3.1, 3.2, and 3.3) to model cloud computing systems and cyber threats associated with them.

We take a generic approach to instantiate our ontologies using AllegroGraph [1] as our back-end data store and analytic platform to perform various querying and reasoning tasks on any generated ontology knowledge base (OKBs) using its REST API. We selected AllegroGraph based on its semantic graph database features that fit well with our ontology engineering requirements.

We start by setting up an AllegroGraph (AG) server and its Python client bundles on an Ubuntu 16.04 Linux machine for our experimental environment. Since we have already designed our domain-specific ontologies using the Protege editor [81], we merely re-create

them in Python programs that write them using the RDFS data model (compatible with our selected back-end triple store), while preserving all the semantics specified during our ontologies design phase. A naive approach to creating and populating our designed ontologies on the AllegroGraph database is outlined in Algorithm 1.

Algorithm 1: Ontology Knowledge Base (OKB) Generator Algorithm	
okbGen():	▷ A generic OKB generator
Set up AllegroGraph global parameters	
Create or access a repository	▷ To act as a back-end AllegroGraph container
Establish a connection object to the created repository	
for <i>Each of designed ontology</i> do	
	Create resources for all of defined classes along with their semantics
	For each class, create all of its necessary predicates resources and their semantics
	Add these resources to the repository using the connection object
	if <i>Any of defined class can be populated automatically</i> then
	Parse the relevant data source feed(s)
	Define a method to align the parsed data and create statements out of them
	Add these statements to the repository using the connection object
	else
	Define a method to receive relevant data manually
	Create statements out of the received data
	Add these statements to the repository using the connection object
	end
	end
	if <i>there are no more classes to be populated</i> then
	Close the connection object
	Shut down the repository access
	else
	Return the connection object
	end
	end

The implementation of our ontology knowledge base generator Algorithm 1 is straightforward, as long as one

- Understands ontology modeling using RDF 1.1 [102], RDFS [103] and AllegroGraph supported semantics (RDFS and OWL predicates) [59].
- Programming in Python 2.7 (also any other programming language supported by AllegroGraph REST API can be used).

The full implementation of our Algorithm 1, along with all the generated ontology knowledge bases are provided within our COCKATOO toolchain (See Chapter 6). Within the COCKATOO toolchain, we leverage “IKAWAFARM” and “LEGOS” tools to generate the knowledge bases pertinent to a cloud system deployment and cyber threat data feeds. In the next subsections, we provide additional details relevant to our example IT systems.

3.3.1. OpenStack Based Cloud System – OKB

In this section, we present one of the open-source cloud projects (**OpenStack**) that enables an IaaS cloud model deployment which can be customized to support other orchestration models (PaaS, & SaaS) as an example instance of our ontologies. OpenStack is “a cloud operating system that controls large pools of compute, storage, and networking resources in a datacenter, and managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface” [95].

We selected OpenStack as our running example for this work since it is an open-source software system and its flexible architecture that is similar to those that are found in most public and private cloud deployments. Here we illustrate how we create a knowledge base for the OpenStack deployment using our ontologies. We extend base classes and populate the classes of the basic cloud computing system ontology (See. Fig. 3.1) using the generic host ontology (See. Fig. 3.2).

Fig. 3.6, 3.7, and 3.9 (or Fig. 3.8) show examples of our OpenStack deployments captured via our cloud computing system and generic host ontology models. In this study, we partially represented the knowledge involved in deploying a basic OpenStack (i.e., Newton

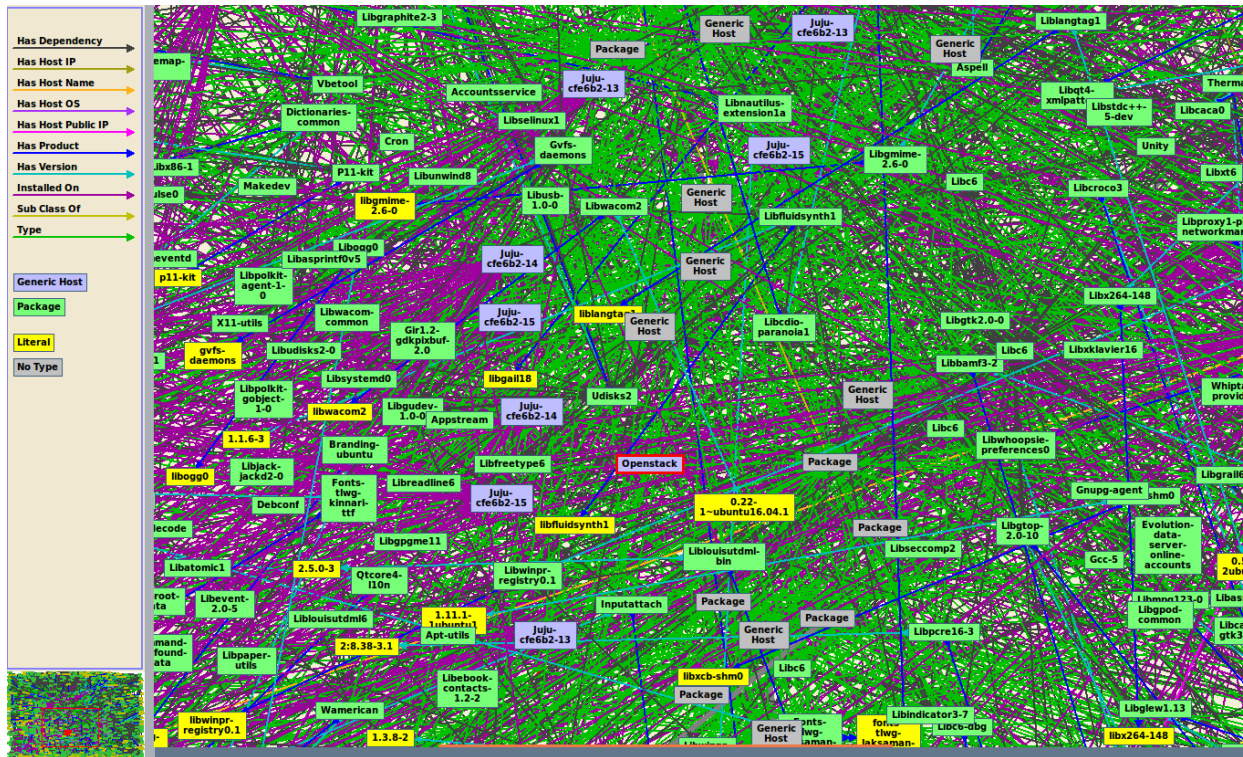


FIGURE 3.6. OpenStack Deployment - Hosts Ontology Knowledge Base

release [94, 2]) powered private cloud with an IaaS offering in one large Virtual machine, running a number of LXD [68] containers.

The generated OpenStack ontology knowledge base (OKB) is stored in an Allegro-Graph repository with these characteristics:

- 10 classes (e.g., Host, ITProduct, Service, etc.)
- 13 predicates (e.g., firstOrderDependency, serviceDependency, hasServiceName, installedOn, etc.)
- 17293 statements (e.g., “lvm2, **installedOn**, juju-df7b65-15”, etc.)

This OpenStack cloud system OKB represents the details about various IT products that are included in deploying OpenStack core services [96] on this private or public cloud [2] such as:

- Identity Service – provides a single point of integration for managing authentication, authorization, and a catalog of services

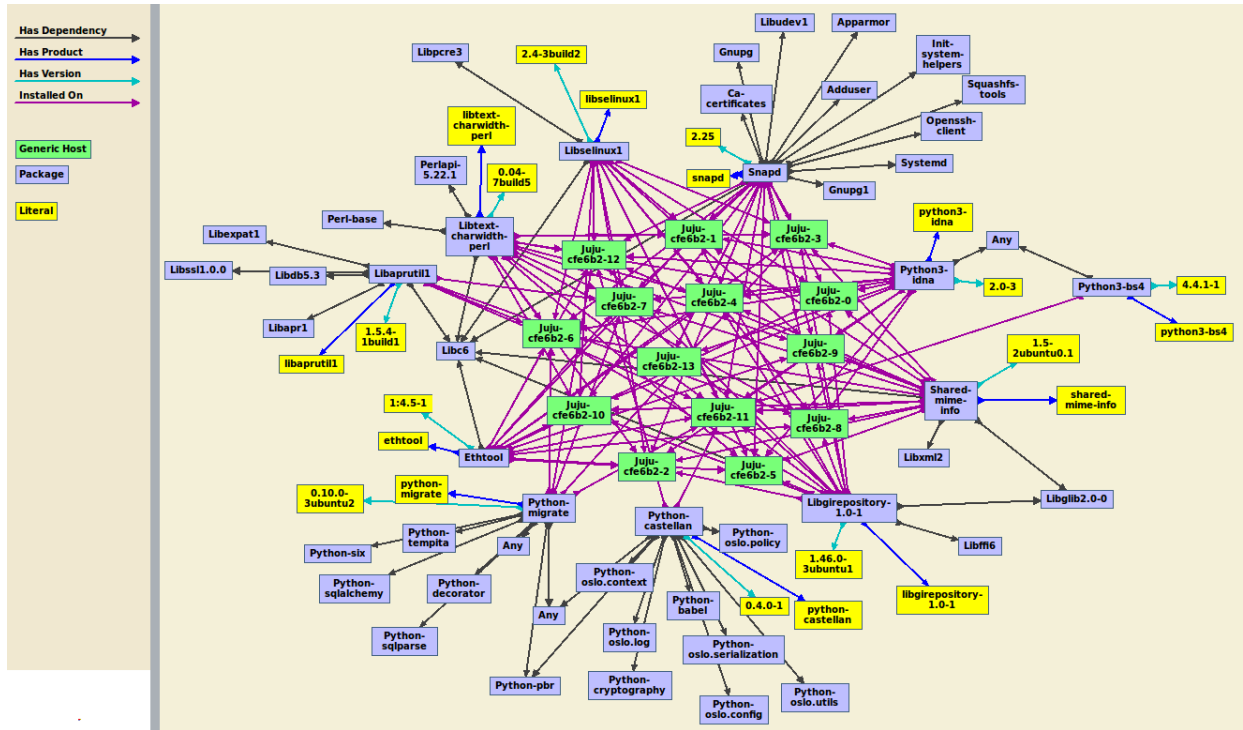


FIGURE 3.7. OpenStack Deployment - Hosts Ontology Knowledge Base – Sample

- Image Service – enables users to discover, register, and retrieve virtual machine images
- Compute Service – is used to host and manage cloud computing systems
- Networking service – allows you to create and attach interface devices managed by other OpenStack services to networks
- Dashboard – is a web interface that enables cloud administrators and users to manage various OpenStack resources and services
- Block Storage service – provides block storage devices to guest instances

The knowledge base of this OpenStack (including subsequent releases) cloud system serves as a source of cloud configurations (based on leveraged IT products to deploy a proof of concept IaaS powered private cloud) to be used as a running example throughout this dissertation.

With the example shown in Fig. 3.9 (or Fig. 3.8) exported as an RDF/XML file (i.e., queens.xml) will enable us to illustrate our proposed techniques (presented in the next chap-

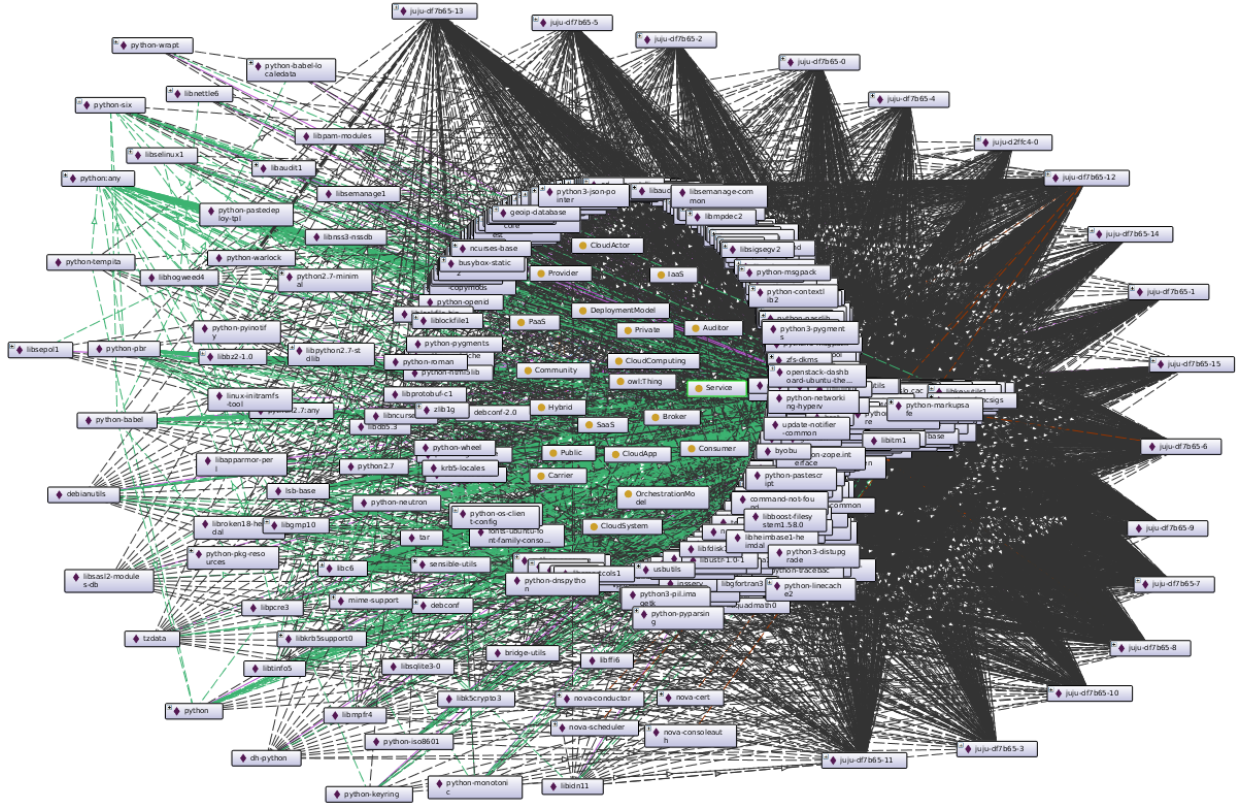


FIGURE 3.8. OpenStack-Netwon Release – Cloud System Ontology Knowledge Base

ters 4 - 6) for assessing and mitigating threats facing the example OpenStack deployment.

3.3.1.1. Other Complex System Deployment Examples – OKBs

In this section, we will use additional examples of some popular products and system. As shown for OpenStack deployment (Sec. 3.3.1), we can use our generic host ontology model to represent the knowledge information used to deploy systems like: “Linux Server [69]”, “Lamp Stack [20]”, and “Kubernetes [12]”. The resulting host knowledge graphs exported in the RDF/XML files of these systems can be used as inputs to other tools presented in later chapters for assessing and mitigating security threats associated with these products.

We illustrate the following system deployments examples for:

- Ubuntu Server deployment in one simple LXD [68] Container machine (See Fig. 3.10 and Fig. 3.11).

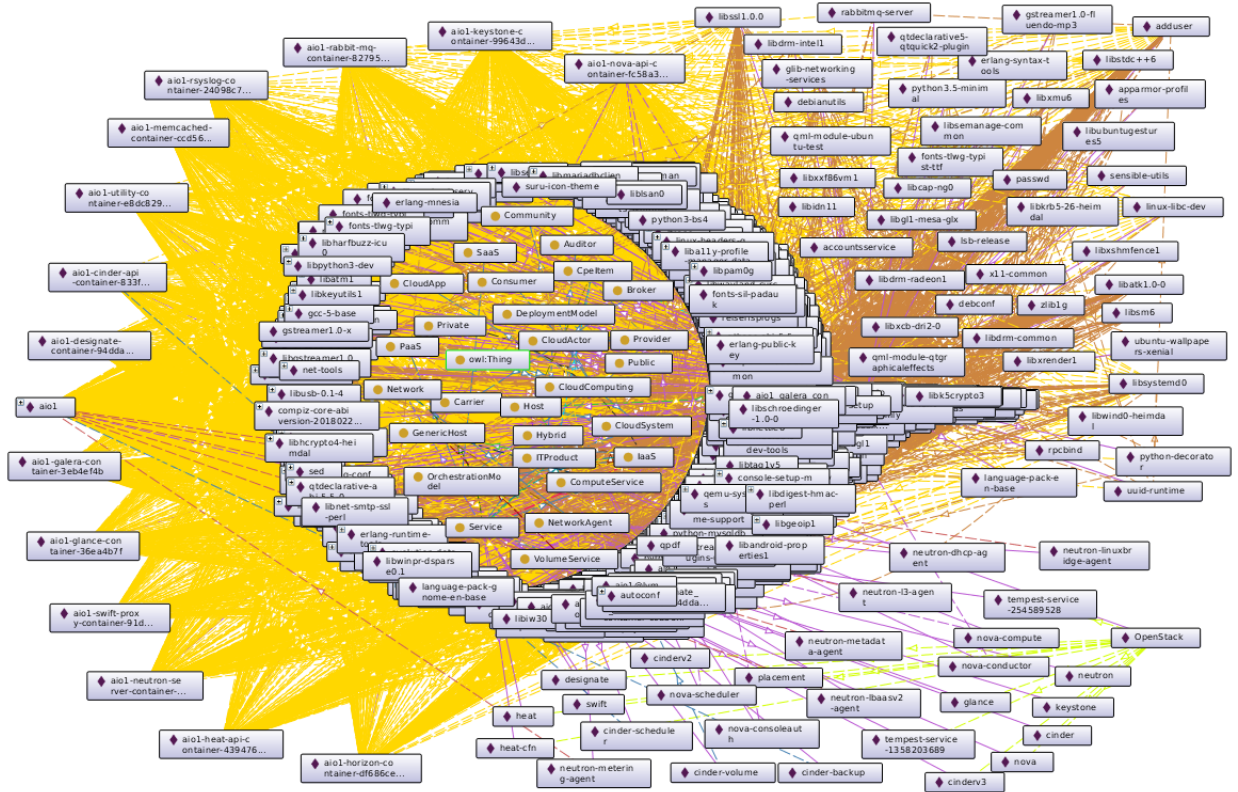


FIGURE 3.9. OpenStack-Queens Release – Cloud System Ontology Knowledge Base

- Lamp Stack deployment as one simple Virtual machine, running an Ubuntu server (See Fig. 3.12 and Fig. 3.13).
- Kubernetes deployment in one large Virtual machine, running a number of LXDC [68] containers (See Fig. 3.14 and Fig. 3.15).

In summary, the host ontology model enables us to represent knowledge about IT products that support any cloud system deployment. Fig. 3.16 shows the number of triples/statements (semantically linked data) about our example systems. From the number of generated triples, we observed that complex systems contain a lot of information (i.e., number of installed packages) compared to small-scale system deployments.

The large volume of captured information and the complexity of our studied systems illustrate the applicability and value of using ontologies to represent information in a machine-readable and semantically defined format for various post-processing tasks. In our

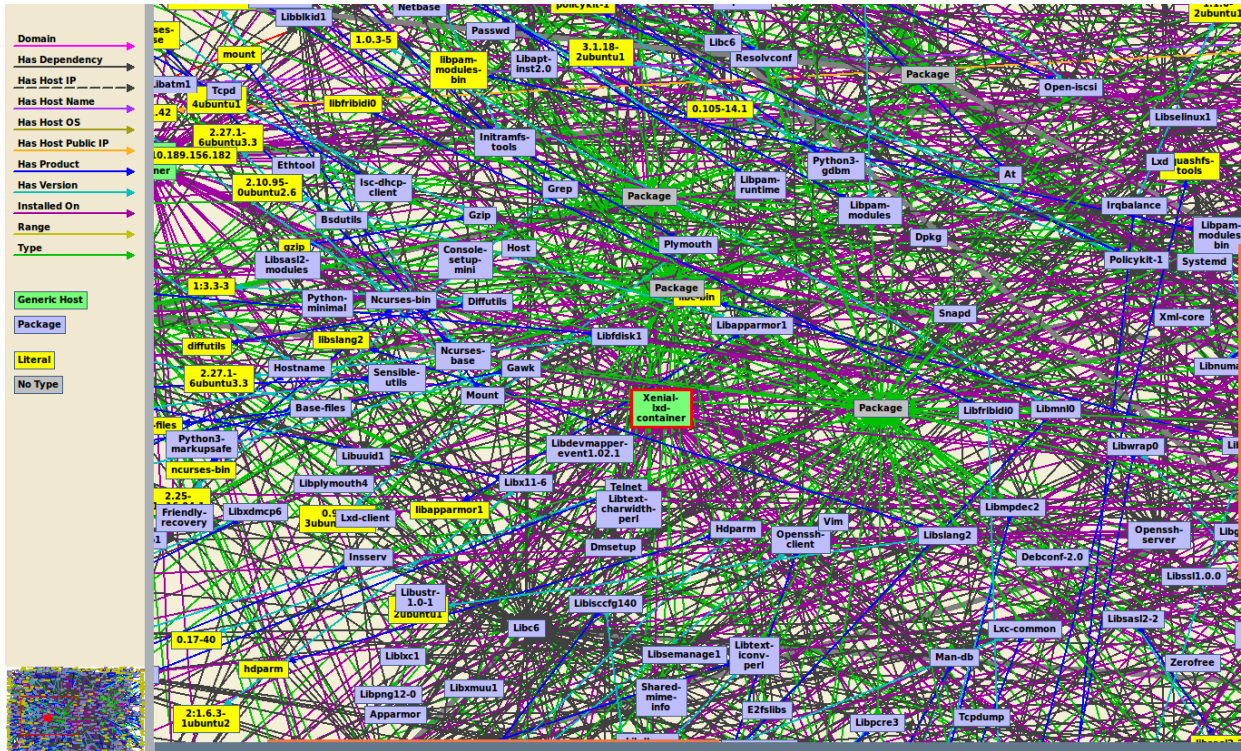


FIGURE 3.10. Ubuntu Server Deployment - Host Ontology Knowledge Base

case, we use the generated knowledge bases as inputs to our other security analysis tools (presented in later chapters).

One can also use a tool such as Gruff [60] to visually build semantic queries to explore the represented knowledge. For example, Fig. 3.17 shows a simple graphical query that allows us to explore the generated host ontology knowledge bases. Using the graphical query shown in Fig. 3.17 we can explore a sample of the generated knowledge graph of our OpenStack deployment from Fig. 3.6 to a simpler version of it in Fig. 3.7 (“Limit” parameter can be adjusted to a desired number of triples to visualize, e.g., Limit=400).

3.3.2. Cyber Threat Data – OKB

The main goal of our work is to capture cyber threat information available in various data feeds using an ontology as described previously. Here we instantiate our ontology using existing source feeds (NVD [85], and ExploitDB [104]). However, our ontology can be easily extended to use data from other sources.

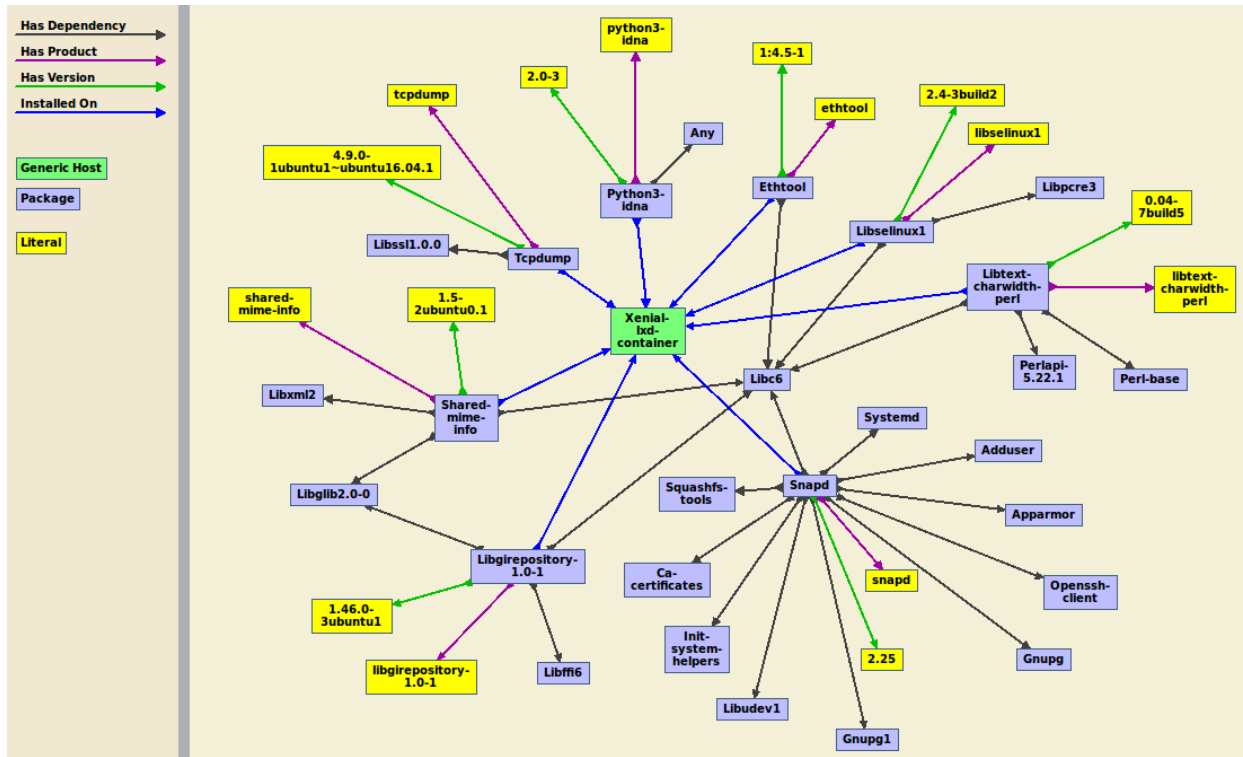


FIGURE 3.11. Ubuntu Server Deployment - Host Ontology Knowledge Base – Sample

Fig. 3.18 and Fig. 3.19 show sample graphs that represent a small set of the generated ontology knowledge base for the cyber threat data domain.

The current generated example cyber threat data OKB is stored in an AllegroGraph repository with these characteristics:

- 11 classes (e.g., Vulnerability, Nvd, CpeItem, etc.)
- 58 predicates (e.g., isAffectedBy, hasCvssAccessVector, hasVulnCwe, etc.)
- 13 millions unique statements (e.g., “CVE-2016-6662, **hasCvssAccessVector**, NETWORK”, etc.)

To broaden our cyber threat data source feeds, we have also integrated one third-party data feed from the McAfee Threat Landscape Dashboard (TLD) [71]. The TLD service highlights the recent top 10 threats observed in critical categories, Threats, Exploit Kits, Campaigns, Ransomware, and Vulnerabilities.

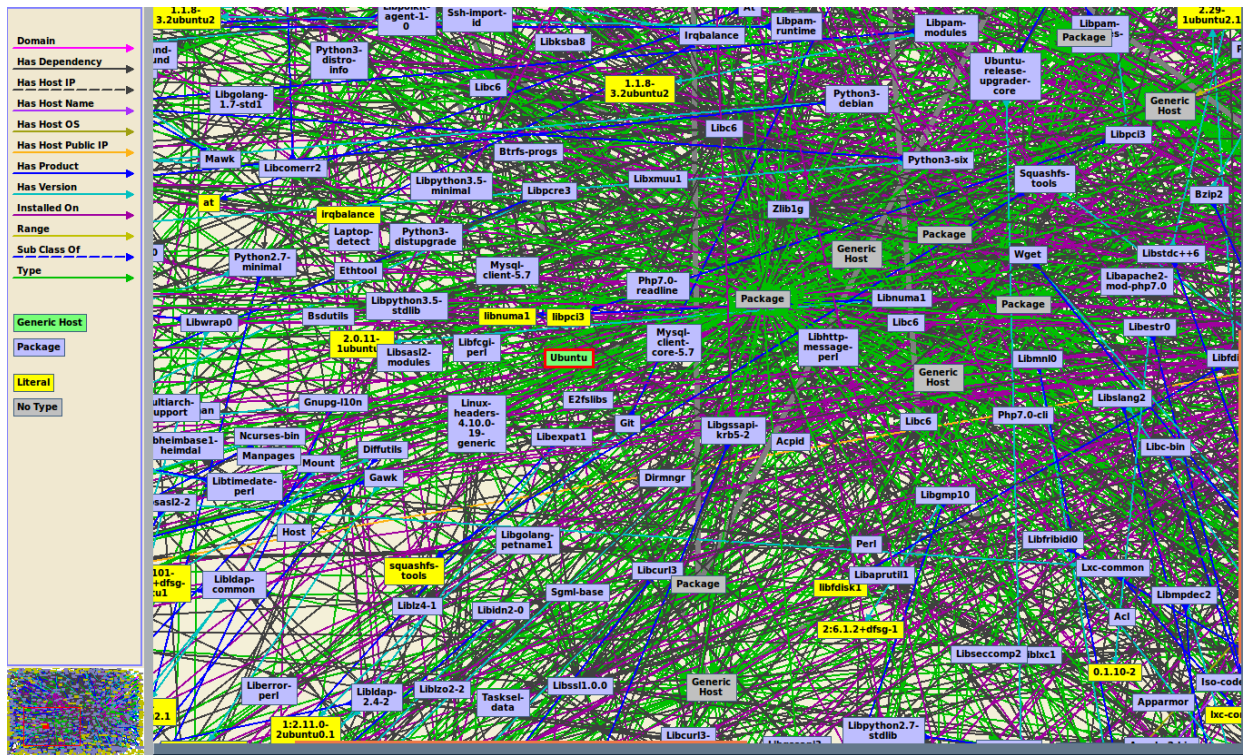


FIGURE 3.12. LAMP Stack Deployment - Host Ontology Knowledge Base

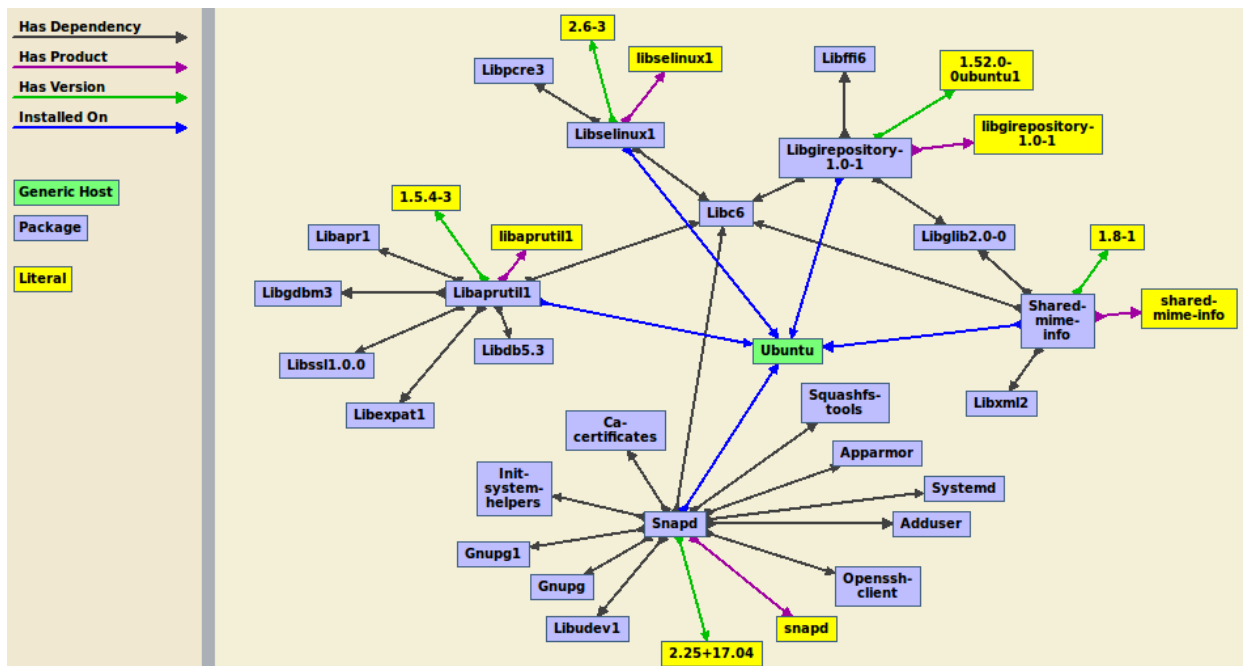


FIGURE 3.13. LAMP Stack Deployment - Host Ontology Knowledge Base – Sample

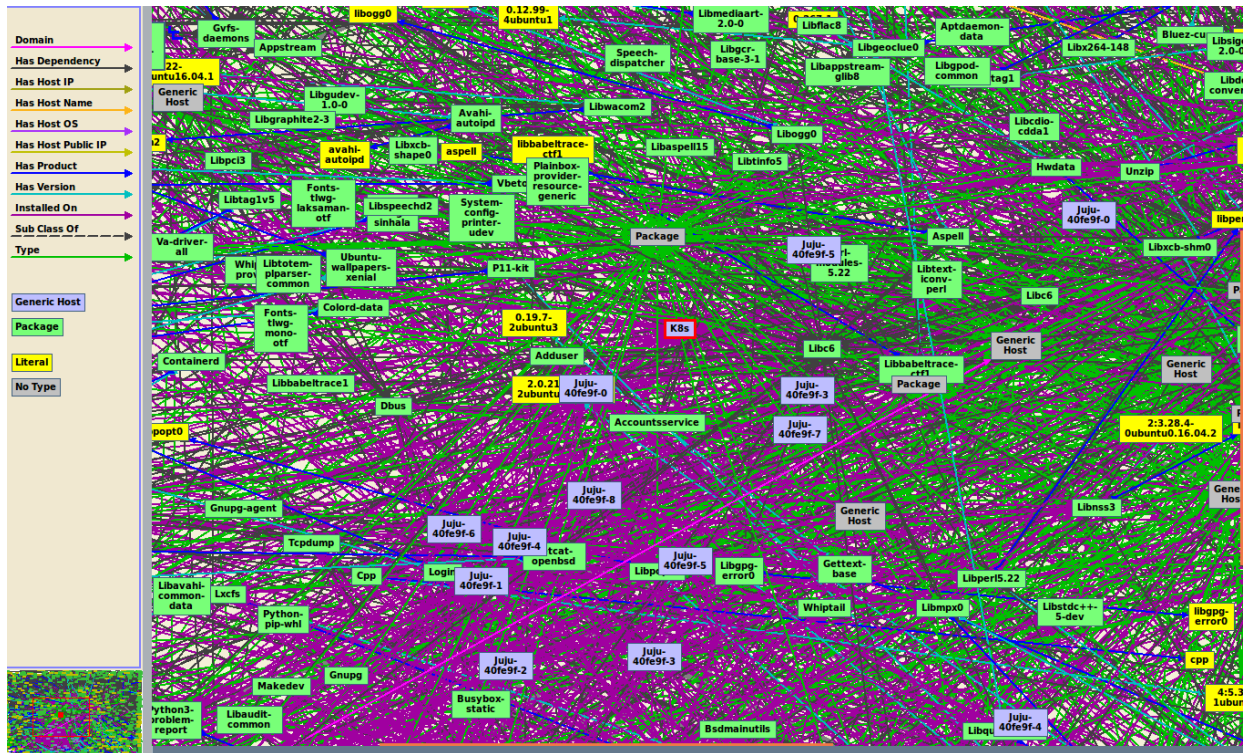


FIGURE 3.14. Kubernetes Deployment - Host Ontology Knowledge Base

Fig. 3.20 illustrates a high-level view of how we are leveraging the McAfee TLD service. We represent the TLD data feed by defining classes to model each threat category conceptually and instances for the top 10 categories. The TLD service uses the Vulnerability identifier as the anchor for other cyber threats. This approach is similar to our core ontology design as shown in our Cyber Threat Data ontology (See Fig. 3.3) and seamlessly integrates with our STIX ontology (See Fig. 3.5).

Using our ontology knowledge base generator Algorithm 1, we extracted information from McAfee’s TLD service, and populated a TLD ontology model, and stored the generated ontology knowledge base (OKB) graph in our back-end AllegroGraph repository which can be queried in a standalone fashion or via a federated query using our IKAWAFARM querying API (See Section 3.4.1). The generated TLD OKB is shown in Fig. 3.21, Fig. 3.22, & Fig. 3.23.

- Since our Cyber Threat Data OKB has information about all currently published and known vulnerabilities, for any query that matches a vulnerability identifier

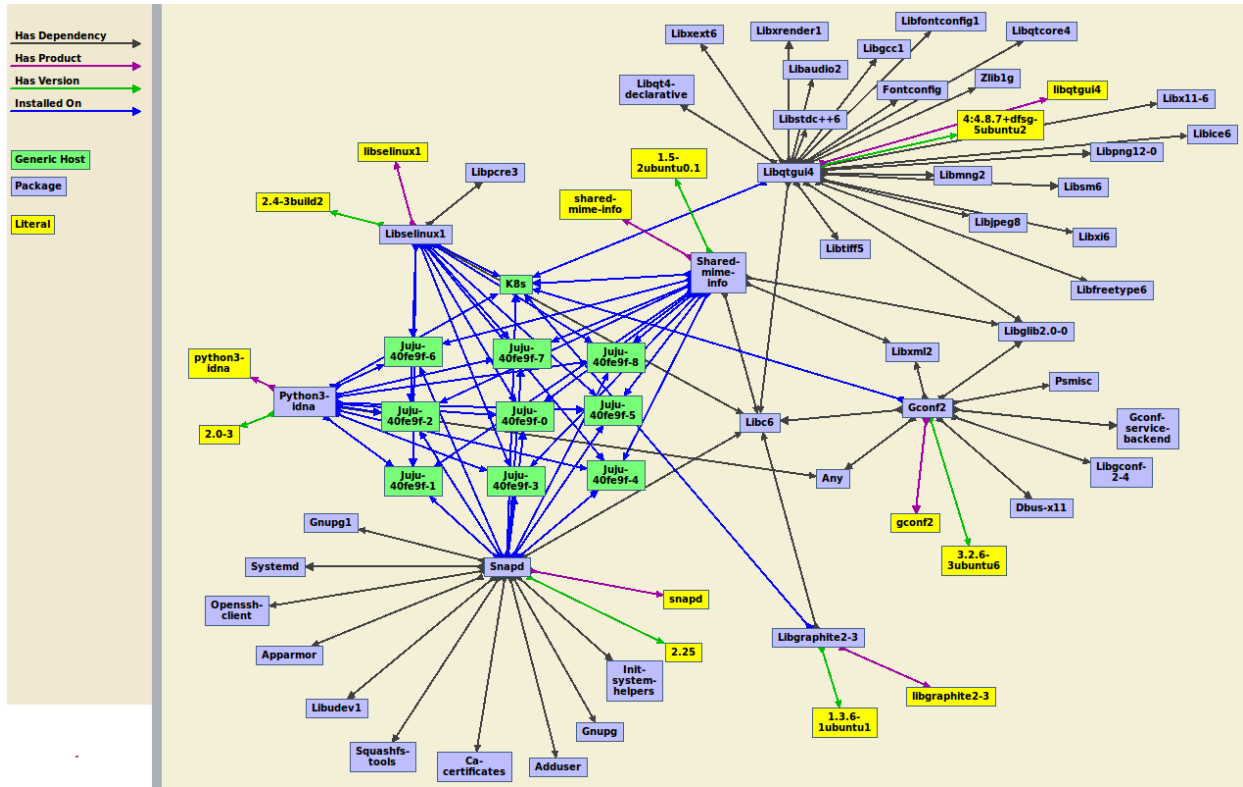


FIGURE 3.15. Kubernetes Deployment - Host Ontology Knowledge Base – Sample

we can infer additional threat information from the TLD OKB and return richer contextual results.

- For resources within the TLD OKB that are limited in expressivity, our Cyber Threat Data OKB can now provide missing links.
- We plan to continue enriching our Cyber Threat Data OKB with other applicable publicly available threat data feeds using our ontological approach.

With these generated ontology knowledge bases, there is a wide range of real-world applications that can be designed using them. In sections 3.4 and 3.5, we present our tools and some basic query operations that can be used to maintain, access or infer information from our ontology knowledge bases (OKBs).

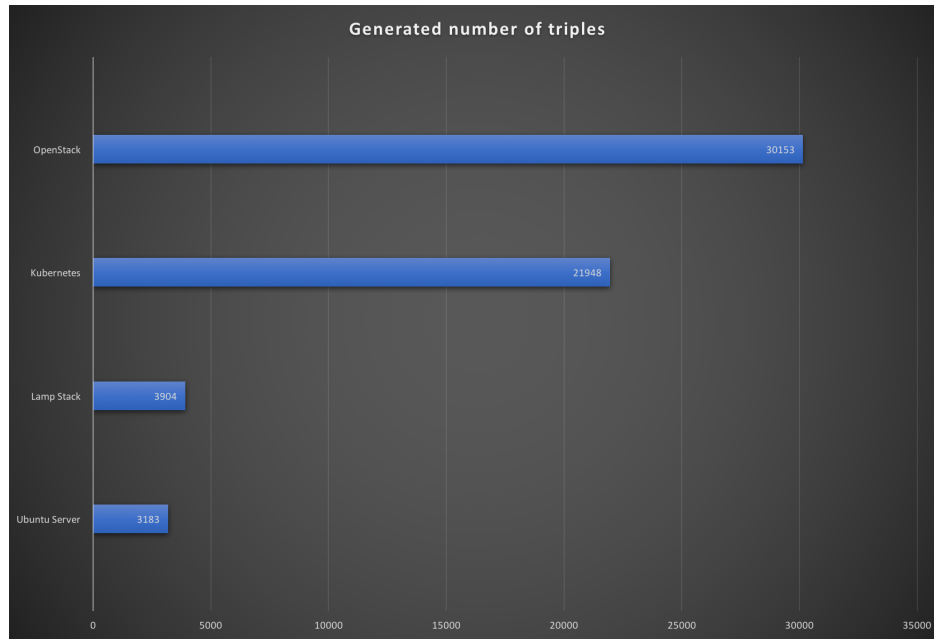


FIGURE 3.16. Statistic Summary of Our Studied and Deployed Systems's - Auto Generated Host Ontology Knowledge Bases

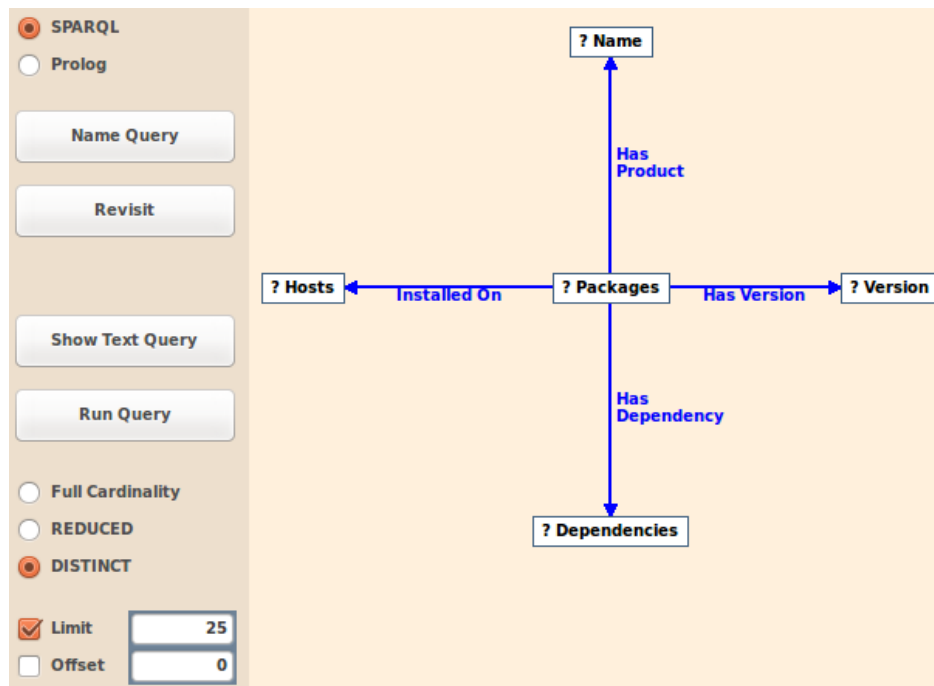


FIGURE 3.17. Host Ontology Knowledge Base - Graphical SPARQL Query Example

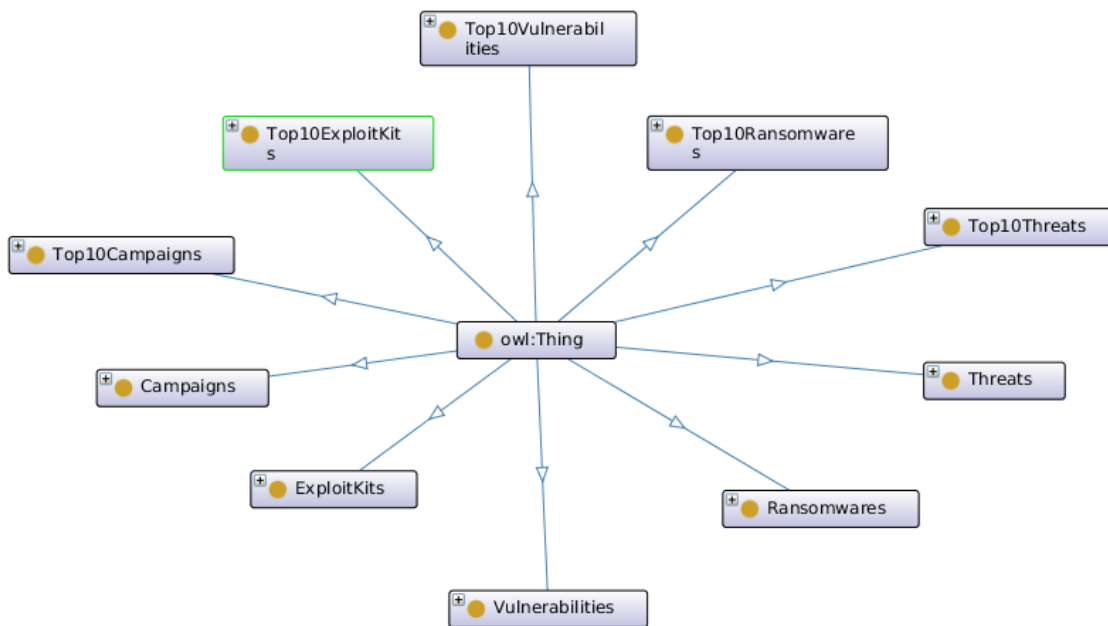


FIGURE 3.20. McAfee TLD Ontology – High Level View

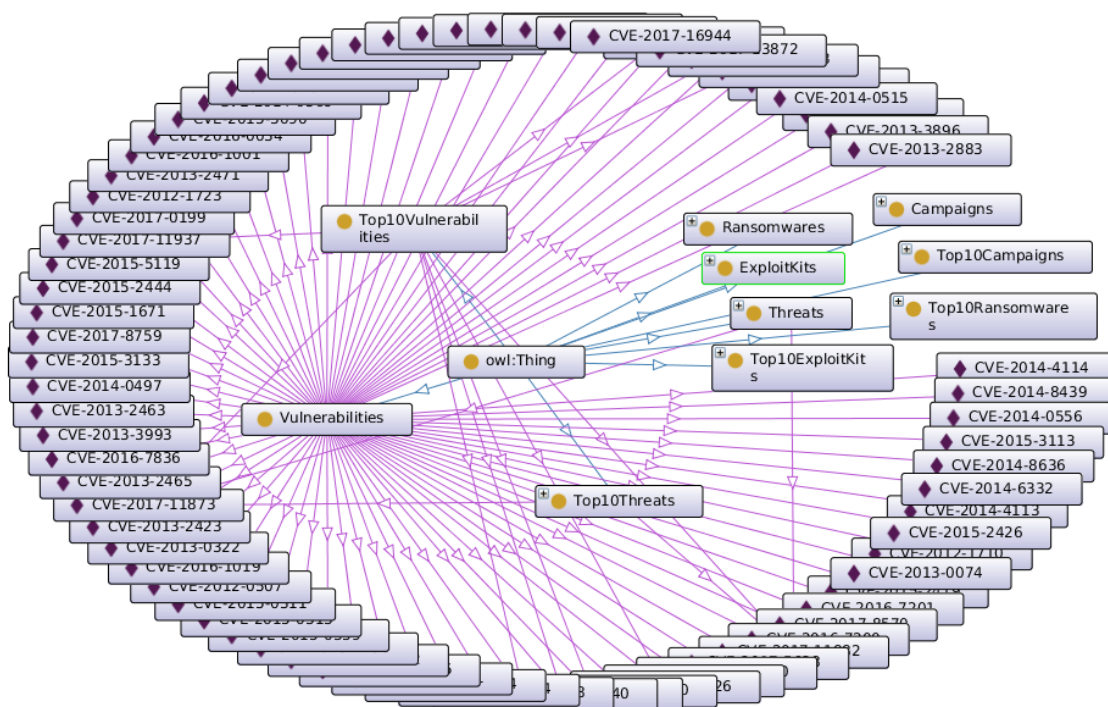


FIGURE 3.21. McAfee TLD Ontology Knowledge Base – Vulnerabilities Instances View

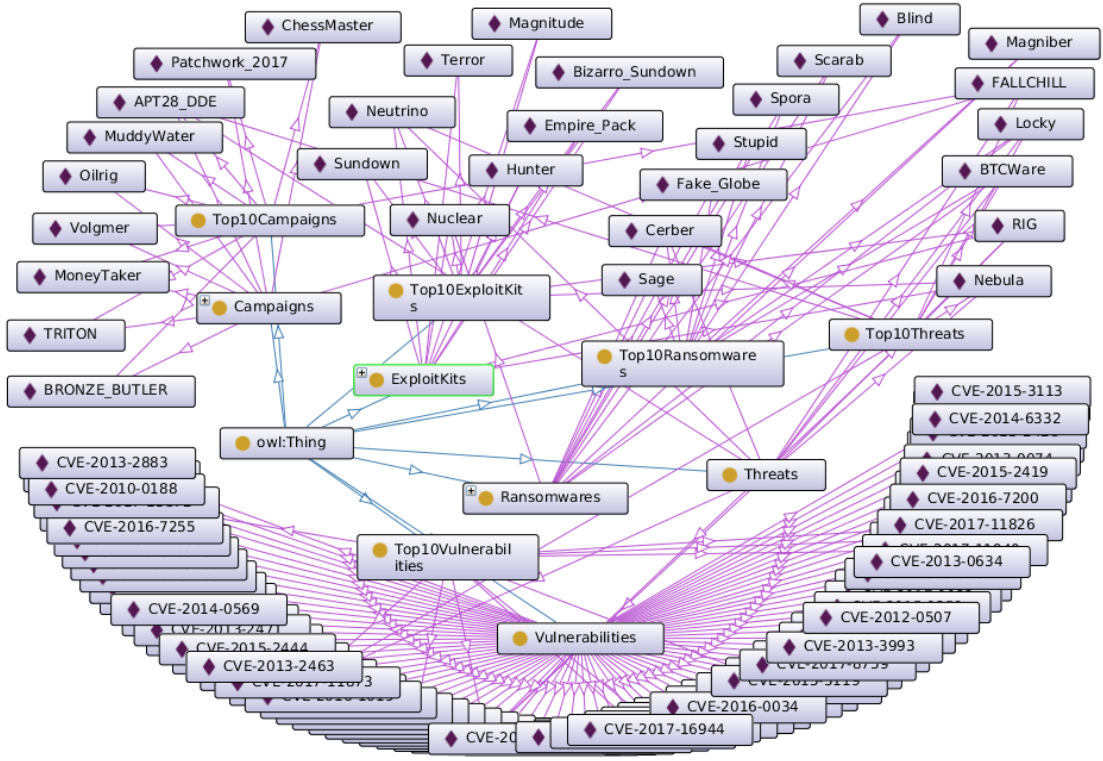


FIGURE 3.22. McAfee TLD Ontology Knowledge Base – Sample View

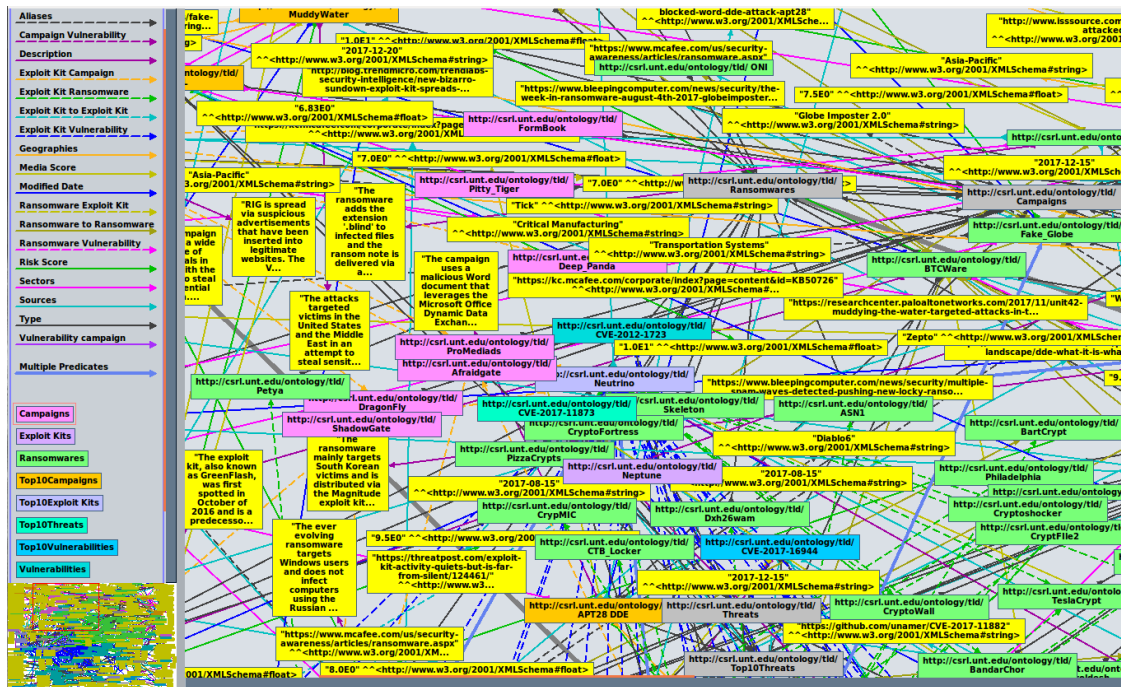


FIGURE 3.23. McAfee TLD Ontology Knowledge Base – Extended View

3.4. Tools for Ontology Engineering

3.4.1. IKAWAFARM

Our ontological engineering approach is captured within a self-contained tool known as “**IKAWAFARM**” which is included within our COCKATOO toolchain (See Chapter 6).

IKAWAFARM permits the creation of scalable ontologies and knowledge bases for cybersecurity threat data and cloud computing domains. It is implemented in Python which can be imported into any Python-based application or system. It provides the following features implemented as modules or standalone applications:

- Ontology design artifacts developed using Protege and AllegroGraph REST API.
- Implementation of our ontology knowledge base generator Algorithm 1.
- Automatic ontology knowledge bases (OKBs) generation for our cyber threat data and cloud system domains.
- API to semantically query the generated OKBs.
- Automation support for collecting various data feeds, setting up a development environment and subsequent interoperability with other toolkits that are a result of this research work.

The IKAWAFARM querying API module implements three types of methods that produce relevant RDF triple/statement results based on:

- (1) SPARQL queries that explore one or more OKB classes or predicates.
- (2) SPARQL queries that perform a direct/indirect match of one or more statements based on user-provided inputs
- (3) SPARQL queries with inference support that uses AllegroGraph RDFS++ reasoner

Each of our IKAWAFARM’s method implements SPARQL queries ranging from simple to advanced form that is supported within the SPARQL 1.1 Query Language [101] to compute relevant results. Some examples of the IKAWAFARM querying API are presented in Section 3.5.

3.4.2. LEGOS

LEGOS is a tool to collect various telemetry data for a computer system using an ontology model (i.e., Fig. 3.2).

A preview of the LEGOS web application prototype is shown in Fig. 6.7. LEGOS uses the IKAWAFARM tool as a background service (See Section 3.4.1) and provides a user-friendly web application via three possible choices:

- **Manual** – A guided web interface that enables users to manually provide the configurations of the IT products that are installed on their hosts. LEGOS then produces an ontology knowledge base of the user-provided assets.
- **Agent** – A self-contained Python package that runs on the deployed cloud hosts and LEGOS automatically build an ontology knowledge base of all identified assets.
- **Agentless** – A web-based interface that collects minimal network-based details from a user about their cloud system and LEGOS attempts to build an ontology knowledge base of exploitable assets.

LEGOS outputs an ontology knowledge base RDF/XML file that can be used to gain visibility (i.e., using Protege [81] or Gruff [60]) into ones deployed cloud system. The other primary use of the LEGOS tool is to standardize a format for collecting information about the cloud system configurations needed by our other tools like “VULCAN” [92] & “NEMESIS” [91] for vulnerability and threat assessment tasks (these tools are described in Chapters 4 and 5).

3.5. Basic Query Operations on Our Ontology Knowledge Bases

Through the back-end AllegroGraph database Web View (AGWebView) [58] or REST API, each generated OKB can be accessed or updated. AllegroGraph has several features that enable us to perform various operations on our OKBs. Some of the functionalities (also integrated within our IKAWAFARM toolkit (See Section 3.4.1)) are listed below:

- Browsing available repositories (i.e., Cloud System and Cyber Threat Data OKBs)
- Issuing SPARQL and Prolog queries

- Capturing a query as a web URL for embedding in applications
- Activating RDFS++ reasoning on a repository
- Setting up free-text indexing for a repository

Some IKAWAFARM querying API examples are listed below.

Statements with `hasCveSummary` » as the predicate.

Subject	Object
CVE-2014-0329	"The TELNET service on the ZTE ZXV10 W300 router 2.1.0 has a hardcoded password ending with airocon for the admin account, which allows remote attackers to obtain administrative access by leveraging knowledge of the MAC address characters present at the beginning of the password."
CVE-2015-1480	"ZOHO ManageEngine ServiceDesk Plus (SDP) before 9.0 build 9031 allows remote authenticated users to obtain sensitive ticket information via a (1) getTicketData action to servlet/AJaxServlet or a direct request to (2) swf/flashreport.swf, (3) reports/flash/details.jsp, or (4) reports/CreateReportTable.jsp."
CVE-2014-9632	"The TDI driver (avgtidix.sys) in AVG Internet Security before 2013.3495 Hot Fix 18 and 2015.x before 2015.5315 and Protection before 2015.5315 allows local users to write to arbitrary memory locations, and consequently gain privileges, via a crafted 0x830020f8 IOCTL call."
CVE-2014-4943	"The PPPoL2TP feature in net/l2tp/l2tp_ppp.c in the Linux kernel through 3.15.6 allows local users to gain privileges by leveraging data-structure differences between an l2tp socket and an inet socket."
CVE-2015-1058	"Multiple cross-site scripting (XSS) vulnerabilities in AdaptCMS 3.0.3 allow remote attackers to inject arbitrary web script or HTML via the (1) data[Category][title] parameter to admin/categories/add, (2) data[Field][title] parameter to admin/fields/ajax_fields/, (3) name property in a basicInfo JSON object to admin/tools/create_theme, (4) data[Link][link_title] parameter to admin/links/links/add, or (5) data[ForumTopic][subject] parameter to forums/off-topic/new."
CVE-2015-1545	"The deref_parseCtrl function in servers/slapd/overlays/deref.c in OpenLDAP 2.4.13 through 2.4.40 allows remote attackers to cause a denial of service (NULL pointer dereference and crash) via an empty attribute list in a deref control in a search request."
CVE-2016-8279	"The video driver in Huawei Mate S smartphones with software CRR-TL00 before CRR-TL00C01B362, CRR-UL20 before CRR-UL20C00B362, CRR-CL00 before CRR-CL00C92B362, and CRR-CL20 before CRR-CL20C92B362; P8 smartphones with software GRA-TL00 before GRA-TL00C01B366, GRA-UL00 before GRA-UL00C00B366, GRA-UL10 before GRA-UL10C00B366, and GRA-CL00 before GRA-CL00C92B366; and Honor 6 and Honor 6 Plus smartphones with software before 6.9.16 allows attackers to cause a denial of service (device reboot) via a crafted application."
CVE-2016-3987	"The HTTP server in Trend Micro Password Manager allows remote web servers to execute arbitrary commands via the url parameter to (1) api/openUrlInDefaultBrowser or (2) api/showSB."
CVE-2016-6662	"Oracle MySQL through 5.5.52, 5.6.x through 5.6.33, and 5.7.x through 5.7.15; MariaDB before 5.5.51, 10.0.x before 10.0.27, and 10.1.x before 10.1.17; and Percona Server before 5.5.51-38.1, 5.6.x before 5.6.32-78.0, and 5.7.x before 5.7.14-7 allow local users to create arbitrary configurations and bypass certain protection mechanisms by setting general_log_file to a my.cnf configuration. NOTE: this can be leveraged to execute arbitrary code with root privileges by setting malloc_lib."

FIGURE 3.24. Sample of Statements with HasCveSummary as a Predicate

- A call to method `predicateStatements(param)` with predicate `hasCveSummary` as input parameter, will output all statements that match the predicate name as shown in Fig. 3.24 (which can be limited to a given number, since several thousand statements/triples match this query). This sample output was captured via the AllegroGraph WebView [58] interface for illustration purposes. Each statement/triple represents a vulnerability identifier as its subject linking the given predicate `hasCveSummary`, with the object of a string literal that contains the actual vulnerability description summary. Our ontology representation approach relies on existing, or user-provided data feeds, in this case, NVD [85] repository for vulnerability details, therefore the similarities in our predicates naming scheme and instances auto-population discussed in the previous section 3.3.
- A call to method `classStatementsOn(param)` with reasoning support and predicate `class` as input parameter, for instance, will explore any of our generated OKBs

(i.e., Cyber Threat Data) and return the main classes instances/statements (i.e., CpeItem, Nvd, VendorStatement, ExploitDB, etc.) that were observed during the ontology knowledge base (OKB) generation. In addition, among the returned results, it includes other instances/statements (i.e., Cpe, Vulnerability, CyberThreat-Data) that were inferred via the predicate “*rdfs:subClassOf*” whereas for example, *Vulnerability* is a super class of Nvd class as shown in Fig. 3.3.

- A call to method “*allDependenciesItProductsRec(param)*” with the name of an installed package on one of the cloud system host servers (i.e., “*python3.5*”) as the input parameter, will return names of all packages that “*python3.5*” depends on, traversing the packages recursively.
- A call to method “*cpeItemStrFinderProdVer(param1, param2)*” will return all matched CpeItem class instances/statements, that match param1 and param2 which have a subject that has two objects (param1, param2). For example if these parameters are predicates “*hasCpeItemProduct & hasCpeItemVersion*”, and param1 equals “*mysql*” and param2 equals to “*5.5.52*”, the method will return the “*cpe:/a:oracle:mysql:5.5.52*” CpeItem instance.
- A call to method “*cveIdSeeker(param)*” will return the vulnerability identification that affects a given input parameter (where param refers to any IT product in CPE format). For instance “*cpe:/a:openldap:openldap:2.4.34*” will return these vulnerability identifiers “*CVE-2013-4449, CVE-2014-97-13, and CVE-2015-1545*”.
- A call to method “*servicesAndHostsMapper()*” makes one call to method “*classStatementsResources(param)*” and another to method “*hostsFinderGivenServiceName(param)*” for cloud service and hosts details. It returns a dictionary object of names of orchestrated services within the cloud system environment along with their respective host(s) that manages them (i.e., service:host [“nova”:*“juju-df7b65-12”*, “keystone”:*“juju-df7b65-8”*, “glance”:*“juju-df7b65-12”*, etc].).
- An example of a slightly more complex method “*indexer()*” provided within our IKAWAFARM API module computes a simple index graph shown in Fig. 3.25.

This method leverages a few other methods to connect our two generated OKBs for the Cyber and Cloud domains, showing all the vulnerabilities of services along with known exploits and any mitigation (or patch) information for each vulnerability.

A more detailed set of examples for our querying API methods is provided in a tutorial supplied within our IKAWAFARM toolkit. In addition, within our COCKATOO toolchain, we offer custom graphical query files that can be loaded into a Gruff [60] instance to allow an analyst to visually explore and gain insight into any of our generated ontology knowledge bases.

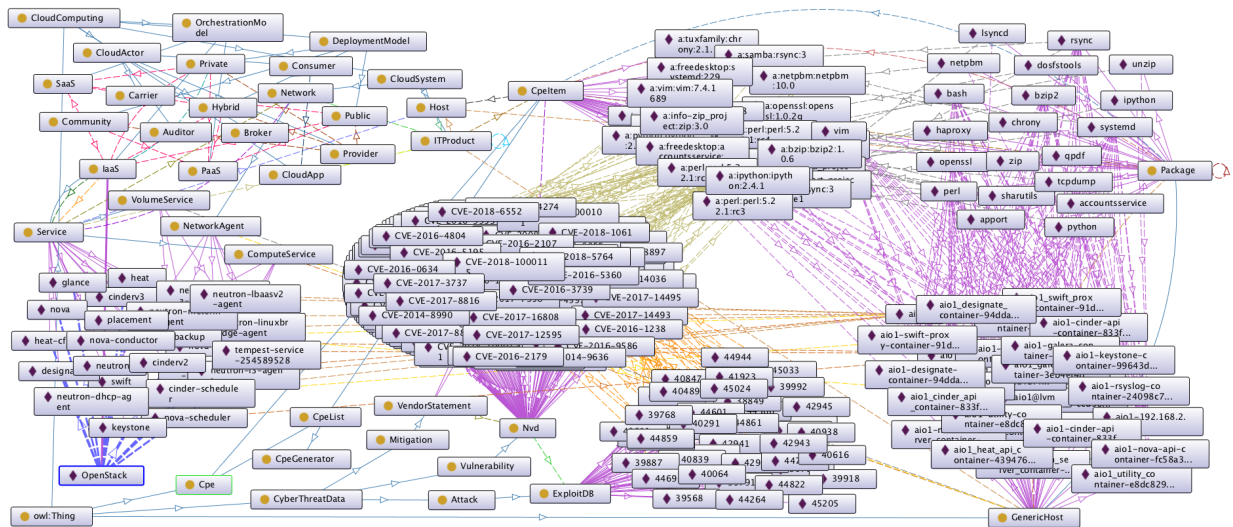


FIGURE 3.25. Indexed Graph – OpenStack Queens Release

CHAPTER 4

ONTOLOGICAL BASED VULNERABILITY ASSESSMENT

4.1. Overview

In Chapter 3, we presented our motivation and ontology engineering approach for modeling cloud computing and cyber threat data domains. In this chapter, we build on our ontology foundation to present our techniques to assess and rank a cloud computing system’s vulnerabilities and predict the number of unknown vulnerabilities in software products.

The next sections (4.2, 4.3, and 4.4) cover our proposed techniques, which have resulted in these peer-reviewed publications and software packages:

- *VULCAN*: Vulnerability assessment framework for cloud computing [92]
- *VULCAN Application*: On-demand vulnerability assessment web application
- A methodology for ranking cloud system vulnerabilities [93]
- Predicting unknown vulnerabilities using software metrics and maturity models [90]
- *SWEEP Tool*: Automate the process of software complexity metrics generation and analysis for any given software product along with its vulnerability history timeline
- *PREDICTION Tool*: A web service to predict the number of unknown vulnerabilities in software products

4.2. Framework to Assess Cloud System Vulnerabilities*

All cloud entities (e.g., orchestrated services, consumer’s applications, and provider’s infrastructure) are exposed to a range of threats from the exploitation of known security vulnerabilities contained in the IT products on which they directly or indirectly depend on. Dependence on shared technologies by cloud services is expected because cloud providers leverage existing solutions to meet customer requirements with Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), or other types of services

*Section 4.2 is reproduced from P. Kamongi, S. Kotikela, K. Kavi, M. Gomathisankaran, and A. Singhal, “VULCAN: Vulnerability Assessment Framework for Cloud Computing,” Proceedings of the Seventh International Conference on Software Security and Reliability (SERE) 2013, Washington, D.C., USA, 18-20 June 2013, with permission from IEEE.

offered by the cloud provider or a third party. This makes the security assessment process very complicated because we need to analyze all the components that are supporting a given service. Yet each cloud actor (e.g., consumer and provider) must be able to ensure the security of their managed cloud system supporting their service(s) offering.

For a given scenario that leverages the cloud fabric, we want to be able to answer these questions for the cloud provider as well as the cloud consumer:

- I have developed and deployed a new cloud product as a service. What is the vulnerability status of its components (during the deployment or operational/maintenance phase)?
- I want to move a deployment or a legacy software application and host it in a new cloud environment. Is there a way to adjust our vulnerability management processes?
- I have designed and deployed a private cloud for my organization. Are there any known or unknown vulnerabilities that may be exploited and exposes our systems and services?
- I am considering adopting a third party software into our production system. Can I find out about vulnerabilities in the software?
- Can I get a daily vulnerability assessment report of our entire deployed cloud system?

To address these essential questions, we developed a framework to assess and manage cloud system vulnerabilities. The codename for our framework is **VULCAN**. The various components comprising VULCAN [92] are illustrated in Fig. 4.1.

The VULCAN architecture leverages these components in a unified fashion:

- IT Products' vulnerability data feeds
- Vulnerability ontology knowledge bases
- Cloud system classifiers
- Vulnerability-based indexing techniques
- Vulnerability index knowledge bases

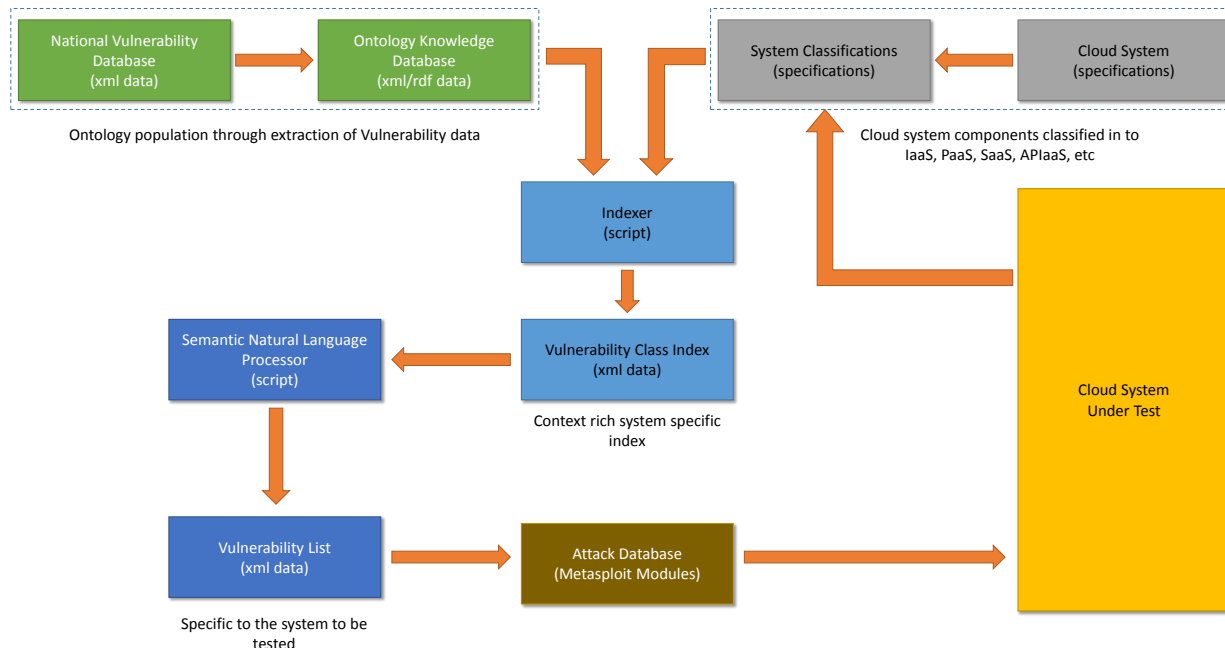


FIGURE 4.1. VULCAN Architecture

- Vulnerability assessment report generator
- Semantic natural language processing

VULCAN components are supported with our middleware and binding scripts which in turn enables us to perform the vulnerability assessment of a given cloud (i.e., Fig. 3.9).

By leveraging the VULCAN framework, we achieve:

- (1) IT Product vulnerabilities modeling
- (2) Automated assessment of vulnerabilities for any cloud computing system
- (3) Discovery of new vulnerabilities from known ones
- (4) Software security penetration helper tool
- (5) Cloud specific vulnerability management platform

In the next Section 4.2.1, we present our VULCAN architectural design in detail.

4.2.1. VULCAN – Architecture

VULCAN uses an ontology-based approach for creating a vulnerability-centric rich cyber threat data knowledge base (See Section 3.2). For assessing vulnerabilities in a specific domain, like cloud computing, we model that domain so that it can seamlessly integrate with

the cyber threat data domain (e.g., information technology (IT) products' vulnerabilities). The modeled domains can be instantiated with relevant data sources and generate a knowledge graph (ontology knowledge base). This knowledge graph can, in turn, be queried and reasoned about the IT system to address various vulnerability issues.

4.2.1.1. IT Products' Vulnerability Data Feeds – NVD

The National Vulnerability Database (NVD) [85] is a SCAP [86] compliant vulnerability database. The NVD database collects vulnerability information from various interrelated vulnerability databases like CVE [76], CWE [77], CPE [83], CVSS [36], and compiles the information into a single database. A unique identifier, CVE or Common Vulnerability Enumeration, tags every entry in the NVD database. This identifier is used to map relevant information for each reported vulnerability along with details found in any of the databases mentioned above. A typical vulnerability entry in the NVD database has the vulnerability identifier, description of the vulnerability, list of software systems and their versions in which this vulnerability is found, and a vulnerability severity score (CVSS) collected from appropriate vulnerability databases. These vulnerability databases are industry standard databases maintained by NIST. Volunteers across the industry contribute to the vulnerability information found in these databases. The SCAP compliance of the NVD database makes it easy to interoperate with other security tools and automate security assessment. The VULCAN framework uses NVD as the source to populate vulnerability information into the cyber threat data ontology knowledge base (presented in Section 3.2.2).

In addition to NVD data feeds (Vulnerability information and Vendors statements), we collect exploits code data feed from Exploit-DB [104] which enables us to find out if any of the NVD published vulnerabilities have publicly known proof of concept exploits.

4.2.1.2. Vulnerability Ontology Knowledge Bases

The Vulnerability ontology knowledge base (OKB) is the ontological database of vulnerability information from the NVD database that is semantically linked with other information such as mitigation techniques (vendor provided patches) and exploit codes (various

Exploit Databases). The Vulnerability OKB is presented in the previous Chapter 3 (See Section 3.3).

4.2.1.3. Cloud System Classifiers

System Classifiers are dynamic inputs provided to the Indexer module, which will create vulnerability classes using a back-end cyber threat data ontology knowledge base.

An example classification includes various vendors in the cloud computing domain and different software or hardware components in each service level of cloud computing services. As shown in Fig. 4.1, cloud computing domains are classified as IaaS, PaaS, or SaaS subdomains. In each of these subdomains, we will include software and hardware components found in cloud computing vendor systems like the Xen hypervisor in the IaaS sub-domain, Google App Engine in the PaaS sub-domain and Salesforce CRM in the SaaS sub-domain. We provide the system classifiers to various levels of detail depending on the interest of the user. The indexer takes these system classifiers as input and recursively crawls through the cyber threat data ontology knowledge base and creates an index. The index consists of vulnerabilities grouped according to the system classifiers provided. The changes in software or hardware in any domain or from any vendor would require updating the system classifiers and re-indexing the ontology knowledge base.

In the previous Chapter (Section 3.2.1), we presented our cloud system modeling approach which can be instantiated to define and create a cloud system classifier for any public/hybrid/private cloud deployment. We developed a helper tool named “LEGOS” (See Section 3.4.2) which enables any cloud actor to bootstrap a representation model of their deployed system. LEGOS generates an ontology knowledge base (OKB) that captures all participating components of the user deployed cloud system (i.e., host software and hardware stack across the entire deployment). This LEGOS generated OKB can be directly fed to the Indexer module for vulnerability classification and indexing.

4.2.1.4. Vulnerability Based Indexing Techniques

The indexer is the software responsible for crawling through the cyber threat data ontology knowledge base to create an index. This index will, in turn, be used by the SNLP (Semantic Natural Language Processor) module to search the ontology knowledge base in response to a user query. The indexer is re-executed every time the ontology knowledge base or system classifiers change. The indexer identifies all the vulnerabilities that are related to the set of software or hardware components listed in the system classifiers and groups them accordingly in the index.

The Indexer module implements various indexing schemes for different usages (e.g., Vulnerability assessment, Threat modeling, and Risk estimation). For instance, the VULCAN Web Application (described in Section 4.2.2.4) leverages the Indexer to satisfy different levels of vulnerability indexing. The currently supported levels of the vulnerability indexing capabilities are listed below.

- *Level 1*: Indexes identifiers of any known vulnerability, exploit, and patch for each given IT asset
- *Level 2*: Indexes identifiers of any known vulnerability for all related products for each given IT asset
- *Level 3*: Extends the *Level 1* indexing technique to provide all key information for each indexed identifier
- *Level 4*: Extends the *Level 3* indexing technique to include related products that have a newer release version for each indexed IT asset
- *Level 5*: Extends the *Level 1* indexing technique to provide all the information that the above indexing levels do provide

In this study, we mainly use Indexes *Level 1-3* for vulnerability assessment tasks and *Level 3-4* indexes for threat modeling and risk assessment of a cloud system. *Level 5* indexes are used for answering user's question about the security posture of their IT systems.

4.2.1.5. Vulnerability Index Knowledge Bases

The Vulnerability Index is the list of all vulnerabilities (encoded as triples) grouped into the categories provided by the system classifiers. These groups are called as “Vulnerability Classes”. Vulnerability classes will assist users in searching for vulnerabilities within a specific domain or sub-domain.

An abstracted index example is shown in Fig. 4.1. The Cloud computing class is at the top level. It has a subclass called PaaS, and the PaaS class has Xen hypervisor as its subclass. In the Xen class, we have the list of vulnerabilities extracted by the indexer from the cyber threat data ontology knowledge base.

Another index example is of a deployed OpenStack cloud system that is shown in Fig. 3.25. Fig. 3.25 illustrates the cloud definition model with a focus on an OpenStack system deployment instance. It shows all orchestrated services and their hosts’ dependencies. For each deployed host, the index shows a sample of installed applications including their Common Platform Enumeration (CPE) representation. In addition, for each IT product (i.e., application), the index shows information about the found vulnerabilities that affect them, and for each vulnerability, information about its exploit and mitigation is represented as well. The index example illustrated in Fig. 3.25 contains plenty of information needed to answer questions about the security posture of the assessed OpenStack deployment.

Each indexing technique described in Section 4.2.1.4 will generate vulnerability classes that augment the cloud system classifier ontology knowledge base of the cloud system under evaluation. In turn, each indexing technique will produce a custom vulnerability indexed knowledge graph (ontology knowledge base). This vulnerability indexed OKB will be used as input for other components of the VULCAN framework to perform vulnerability assessment tasks.

4.2.1.6. Vulnerability Assessment Report Generator

Vulnerability Assessment Report (VAR) generation leverages a Vulnerability Indexed OKB as the source of vulnerability information and the specific components of the assessed cloud system that are affected. The VAR generator can be invoked via the VULCAN Web

Application (See Section 4.2.2.4) to render a vulnerability assessment report to inform the cloud actor. This contains information regarding the vulnerability status of the cloud system along with actionable insights on which IT product needs their attention.

4.2.1.7. Semantic Natural Language Processing (SNLP)

The Semantic Natural Language Processor (Illustrated in Fig. 4.2) enables users and our automated software agents (i.e., web applications and chatbots) to search and reason about vulnerabilities. It includes various sub-components which are capable of pattern matching, keyword search, and reason over properties and relationships of the classes in the ontology knowledge bases. SNLP takes input from a user or a software agent and tries to interpret what the user is asking, and returns a list of vulnerabilities for the requested product or class. SNLP is capable of looking up vulnerabilities for the requested product and listing vulnerabilities in a particular class or product across various vendors. It also can reason and list vulnerabilities for the technology or framework used in the user’s application stack and system deployments.

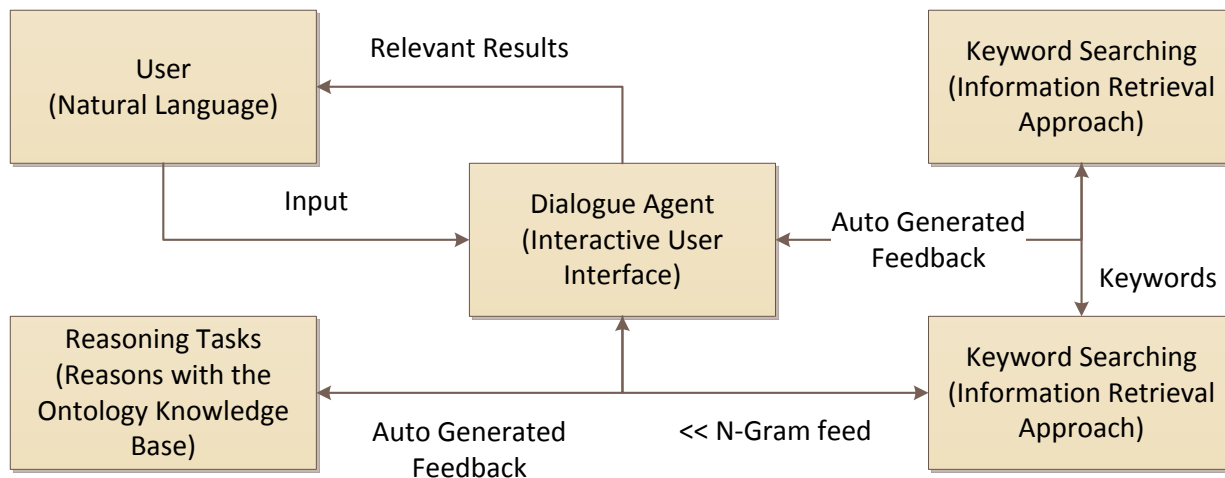


FIGURE 4.2. VULCAN – Semantic Natural Language Processor (SNLP)

4.2.1.8. Other Components

In previous subsections, the main components and modules of VULCAN were presented. The remaining components of the VULCAN architecture shown in Fig. 4.1 are the

customizable features that permit testing of Cloud Systems. The Vulnerability List is generated from the SNLP component after processing natural language queries. Also, the Attack Database is an independent source of attack modules – Metasploit [98] in the current implementation. These attacks can be used to test if a system contains a specific vulnerability and if the attack is successful or not.

In subsection 4.2.2, we present the implementation of VULCAN architecture components and integration of these components in the *VULCAN Web Application*. We use the VULCAN Application, and some use cases to demonstrate the capabilities of the VULCAN framework.

4.2.2. VULCAN - Implementation

Python is used to implement the VULCAN framework since Python is portable across computing platforms. VULCAN components are modular in the sense that they can be changed or replaced with new modules without affecting other components. This also allowed us to build components separately and integrate them in the VULCAN framework (in a sense adopting agile software development practice). One or more components can be invoked from the *VULCAN Web Application* (based on the workflow among the components).

The “*IT Products’ vulnerability data feeds*” and “*Vulnerability ontology knowledge bases*” components of VULCAN are implemented using our “*IKAWAFARM*” tool (presented in Chapter 3, Section 3.4.1).

The “*Cloud system classifiers*” component is implemented using the “*LEGOS*” tool (presented in Chapter 3, Section 3.4.2).

4.2.2.1. Vulnerability Based Indexing Techniques

The “*Vulnerability based indexing techniques*” component is implemented within our VULCAN Python project. As previously presented, our VULCAN framework offers various vulnerability indexing types to satisfy different needs.

Within our VULCAN Python project, we implemented each vulnerability index type as a Python module which can be called from any Python application. Each vulnerability

index module leverages our backend cyber threat data ontology knowledge bases via an API that is maintained by our IKAWAFARM tool.

The execution of any vulnerability index type when given an input use case (i.e., a deployed “cloud system - ontology knowledge base” generated by our LEGOS tool), will output a matching “*Vulnerability index knowledge base*” and will be stored within our backend semantic graph database as a repository container. This *Vulnerability Index knowledge base* output can then be exported as an “RDF/XML” file automatically or via the user prompt for post analysis tasks using our VULCAN web application or a compatible visualization tool (i.e., gruff [60], protege [81]).

4.2.2.2. Vulnerability Assessment Report Generator

The “*Vulnerability assessment report generator*” component is implemented and integrated within our *VULCAN Web Application*. We used the Django framework [38] templating feature for generating a vulnerability assessment report (VAR) template. The template is made of these key table of contents views:

- *Executive Summary* – provides a brief summary of known vulnerabilities that affect the assessed IT system (e.g., “Deployed cloud system”) in terms of the number of affected hosts, installed software systems on each host and the severity of the found vulnerabilities (e.g., how many vulnerabilities have publicly known exploits and how many can be exploited via a network attack vector).
- *Asset View* – enables the user to explore each affected application/os/hardware/host(s) of the assessed IT system. For instance, by navigating through a host pivot point, a list of assessed host identifiers will be presented where each host identifier enables the user to view all affected software products (e.g., Operating Systems, Applications, and Packages), then by selecting a given product, the web view will expand to provide details about all found vulnerabilities.
- *Vulnerability View* – provides an explorable list of discovered vulnerability identifiers in CVE format [76]. For each vulnerability identifier, the user can view the complete details on that particular vulnerability, along with associated references to threat

data feeds (e.g., NVD) and subsequently to how that particular vulnerability can be exploited, mitigated or patched.

- *Exploit view* – provides a list of identified publicly known exploit identifiers (from Exploit-DB [104]) that can be leveraged to exploit some of the known vulnerabilities that affect the assessed IT system. At present we are limiting our sources to very few exploit databases and thus are limited to the number of known exploits.
- *Mitigation View* – provides a list of known public fixes to mitigate some of the found vulnerabilities. We are limited by the publicly known information or information made available by the vendor in this connection.
- *Visualization View* – provide simple Gruff [60] graphical queries that an analyst can use to explore and infer insights from the VULCAN generated vulnerability index ontology knowledge base for the assessed IT system.
- *Next Steps* – provides some recommendations to take based on the vulnerability assessment findings

In response to user tasks for generating different vulnerability index (VI) knowledge graphs and reports, the VULCAN application agent will invoke custom APIs to find all the relevant information from the vulnerability index and populates the VULCAN-App templates so that this user can navigate and explore the contents of each report.

4.2.2.3. Semantic Natural Language Processing

The “*Semantic natural language processing*” component is implemented and integrated within our *VULCAN Web Application* via the *Query* window where a user can initiate a chatbot session. At present, we have implemented some common scenarios of user queries and tested the responses.

We have also implemented some components/modules to drive real-life use cases, where our VULCAN framework can address common routine tasks that a security analyst may encounter on a daily basis.

4.2.2.4. VULCAN Web Application

We designed and implemented a prototype “*VULCAN Web Application*” to demonstrate various use cases of our Vulcan framework. The primary VULCAN-app user interface is shown in Fig. 6.9. The application features are divided into three columns in the window: Assess, Report, and Query.

- The **Assess** category of the application empowers the user to:
 - Generate an ontology knowledge base (OKB) of their deployed cloud system using our LEGOS tool
 - Task VULCAN-App to index the user provided OKB (or generated using LEGOS) of their system (choices for different levels of vulnerability indexing are presented to the user during the process)
 - Schedule the vulnerability indexing task periodically since our back-end cyber threat data ontology knowledge base will evolve on a regular basis based on updates to the threat data feeds
- The **Report** category of the application enables the user to:
 - Upload a previously generated vulnerability index ontology knowledge base of a target system (VULCAN-App has a command-line interface that offers features similar to the web interface)
 - Select one of the vulnerability indexes generated from the “Assess” category
 - Render and explore a vulnerability assessment report on-demand by selecting the tasked vulnerability index ontology knowledge base
- The **Query** category of the application enables the user to:
 - Initiate a chatbot session which uses a conversation agent that can understand and interpret user-generated vulnerability assessment reports
 - Ask a question in English to converse with the chatbot
 - Define a daily workflow for users to maintain the security postures of their systems.

We implemented the VULCAN web application using third-party software packages

including:

- Django web framework [38]
- Bootstrap framework [21]
- Protege: an ontology editor and framework for building intelligent systems [81]
- AllegroGraph semantic graph database [1]
- Gruff: a graphical triple-store browser for AllegroGraph [60]

Also, we developed various other tools needed for VULCAN that include (as reported in the previous chapter) :

- LEGOS tool
- IKAWAFARM tool
- Vulnerability indexing modules
- Vulnerability assessment report generator
- Vulnerability information chatbot

The original publication on the VULCAN Framework [92] described the goals of VULCAN without complete implementation of various components. We now have a prototype implementation of VULCAN as well as other systems in our COCKATOO toolchain (See Chapter 6).

4.2.3. VULCAN - Evaluation

To test the VULCAN framework, we evaluated several IT systems. Here we illustrate how VULCAN is tested on an OpenStack cloud deployment (See Section 3.3.1).

We deployed OpenStack powered private cloud systems using the Conjure-up [2] and OpenStack-Ansible [97] orchestration tools. An overview of one of our deployed OpenStack cloud systems is shown in Fig. 3.9 (or Fig. 3.8). Fig. 3.9 was generated using the Protege [81] editor activated with the ontology knowledge base of the deployed OpenStack cloud system, which in turn was generated by our LEGOS tool (See Section 3.4.2). A LEGOS agent was installed in each of the containers that were bootstrapped on the OpenStack system and collected telemetry data for the container including information on all software systems running

in the container. The telemetry data is collected automatically and stored in an AllegroGraph [1] database repository. This information can be stored locally where the VULCAN application is running or on a remote system. The generated AllegroGraph repository can be accessed via an API or exported as an XML/RDF file for post-processing tasks (like visualizations shown in Fig. 3.9 or fed to the VULCAN Web Application for vulnerability assessment).

The main achievement of the VULCAN framework is automation and the scalability of vulnerability assessment.

For our case study of the OpenStack cloud system, we tackled the first challenge of modeling and representing the deployed system using LEGOS agents which alleviated the manual work of inputting information about each software system running in each container.

The second challenge we addressed is the automatic vulnerability assessment of the deployed OpenStack.

- We load the LEGOS generated OKB of the deployed OpenStack system into the VULCAN Web Application (or Command line interface) as shown in Fig. 4.3. Then, we select and task the given cloud system classifier OKB for vulnerability indexing.
- Different vulnerability indexing techniques are presented, and the user can select which one best suits their vulnerability assessment level (Level 1-5). Then, a background process will invoke the Indexer module that will crawl the entire Cyber Threat Data ontology knowledge base and generate a vulnerability index knowledge graph (which will become available within the VULCAN web application dashboard once the indexing process is complete). Fig. 4.4 shows different vulnerability indexes (with Vulnerability Index – Level 3, aka. “Index_Vulcan” highlighted) generated for the selected cloud system OKB.
- Table 4.1 shows a comparison of different statistics generated for each of the supported vulnerability indexing levels (1-4) (See Section 4.2.1.4) for the given OpenStack OKB generated by our LEGOS tool which contained 30153 triples representing information regarding all the deployed IT assets, hosts and installed software pack-

ages. Note that the Vulnerability Index Level 5 statistics are not shown since they are costly to generate for a large-scale deployment such as the OpenStack studied here. We recommend that a Level 5 index be used for smaller systems or a single host. We also recognize the limitation of our indexing capability since we rely on a few open-source data feeds for our Vulnerability, Exploit, and Mitigation information to populate our Cyber Threat Data ontology. However, our framework can be extended to include additional sources of information.

- The user can then explore the vulnerability indexes generated and select which one to use for the vulnerability assessment report generation. For instance, Fig. 4.5 shows an on-the-fly rendered vulnerability report when the “Level 3” vulnerability index is selected. The user can navigate this report to view different levels of details on vulnerabilities. Fig. 4.6 shows the executive summary for the example OpenStack deployment discussed above. Other reports generated are shown in Fig. 4.7 to Fig. 4.15.
- The user can also elect to use the SNLP-based Chatbot feature within the VULCAN web application to find answers to the vulnerability status of their deployed cloud system. Figures 4.16, 4.17, and 4.18 show a simple chatbot user interaction. Advanced features of the VULCAN Web Application include the support of scheduled vulnerability indexing and user-defined routines via the chatbot.

Beyond the VULCAN web application, VULCAN framework has other applications. In subsequent Sections 4.3 and 4.4, we present two examples of our work that leverage VULCAN framework as the source of semantically linked vulnerability information:

- A methodology for ranking cloud system vulnerabilities [93]
- Predicting unknown vulnerabilities using software metrics and maturity models (Viz., SWEEP and PREDICTION modules) [90]

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Did you know that there are #90K+ publicly known vulnerabilities?

Do you know if any of your IT asset is affected?

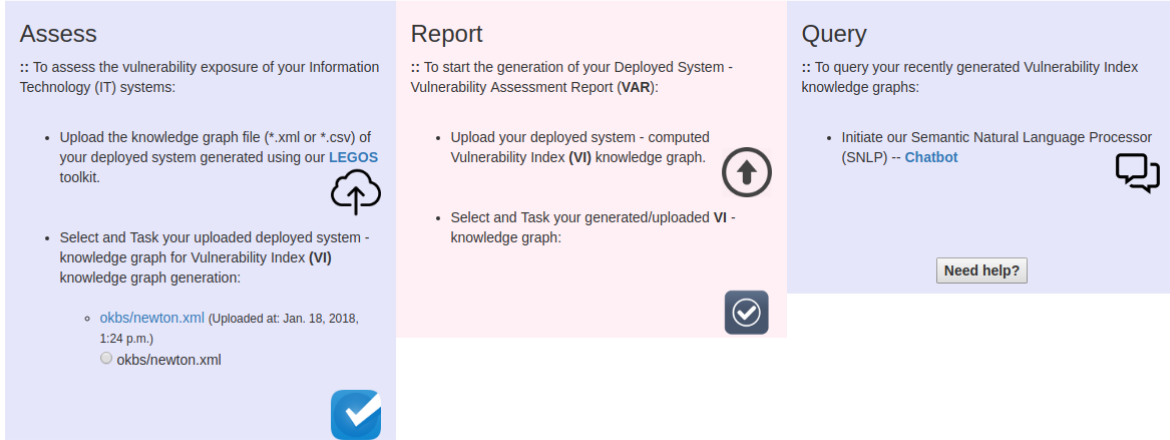


FIGURE 4.3. VULCAN Web Application - OpenStack:Newton Legos Generated OKB Input

Vulnerability Index (VI)	#Triples	#Vulnerabilities	#Exploits	#Mitigations
VI-Level1 (aka. Default)	30388	68	3	0
VI-Level2 (aka. Sweep)	38044	308	0	0
VI-Level3 (aka. Vulcan)	37603	68	3	0
VI-Level4 (aka. Nemesis)	63969	242	7	13
VI-Level5 (aka. Cockatoo)	N/A	N/A	N/A	N/A

TABLE 4.1. VULCAN Web Application - OpenStack:Newton Vulnerability Indexes OKBs Comparison

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems
 Did you know that there are #90K+ publicly known vulnerabilities?
 Do you know if any of your IT asset is affected?

Assess

:: To assess the vulnerability exposure of your Information Technology (IT) systems:

- Upload the knowledge graph file (*.xml or *.csv) of your deployed system generated using our **LEGOS** toolkit.
- Select and Task your uploaded deployed system - knowledge graph for Vulnerability Index (VI) knowledge graph generation:
 - okbs/newton.xml (Uploaded at: Jan. 18, 2018, 1:24 p.m.)
 - okbs/newton.xml

Report

:: To start the generation of your Deployed System - Vulnerability Assessment Report (VAR):

- Upload your deployed system - computed Vulnerability Index (VI) knowledge graph.
- Select and Task your generated/uploaded VI - knowledge graph:
 - documents/newton-Index_Default.xml (Uploaded at: Jan. 18, 2018, 1:43 p.m.)
 - documents/newton-Index_Default.xml
 - documents/newton-Index_Sweep.xml (Uploaded at: Jan. 18, 2018, 1:43 p.m.)
 - documents/newton-Index_Sweep.xml
 - documents/newton-Index_Vulcan.xml (Uploaded at: Jan. 18, 2018, 1:44 p.m.)
 - documents/newton-Index_Vulcan.xml
 - documents/newton-Index_Nemesis.xml (Uploaded at: Jan. 18, 2018, 1:45 p.m.)
 - documents/newton-Index_Nemesis.xml

Query

:: To query your recently generated Vulnerability Index knowledge graphs:

- Initiate our Semantic Natural Language Processor (SNLP) -- [Chatbot](#)

[Need help?](#)

FIGURE 4.4. VULCAN Web Application - OpenStack:Newton Generated Vulnerability Indexes OKBs

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems
 Do you know if any of your IT asset is affected by one of the known vulnerabilities?

VULCAN - Vulnerability Assessment Report (VAR) for **newton-Index_Vulcan.xml** - Input

[Table of Contents](#)

1. Executive Summary
2. Asset View
3. Vulnerability View
4. Exploit view
5. Mitigation View
6. Visualization View
7. Next Steps



FIGURE 4.5. VULCAN Web Application - OpenStack:Newton Vulnerability Assessment Report using a Generated Vulnerability Index Level 3 OKB

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

Executive Summary

From a total of **37603** facts, that we know about your deployed system. Here are our top 5 vulnerability assessment observations:

1. We have assessed **17** of your deployed hosts, and identified **2421** unique installed packages.
Where, **31** candidate packages, including the hosts Operating Systems have **68** publicly known vulnerabilities.
2. Among all **68** publicly known vulnerabilities, **3** of them have public proof of concept exploit codes available.
3. Among all **68** publicly known vulnerabilities, **0** of them have public mitigation patches available.
4. Among all **68** publicly known vulnerabilities, **42** of them can be exploited via a NETWORK attack vector.
5. Among all **68** publicly known vulnerabilities, **26** of them can only be exploited via a LOCAL attack vector.



FIGURE 4.6. VULCAN Web Application - OpenStack:Newton Executive Summary View

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

Assets View

To view related information about each of your **17** deployed hosts, start to pivot from each of these below hostnames:

- juju-cfe6b2-1
- juju-cfe6b2-0
- juju-cfe6b2-3
- juju-cfe6b2-2
- juju-cfe6b2-5
- juju-cfe6b2-4
- juju-cfe6b2-7
- juju-cfe6b2-6
- juju-cfe6b2-9
- juju-cfe6b2-8
- juju-cfe6b2-10
- juju-cfe6b2-11
- openstack
- juju-cfe6b2-13
- juju-cfe6b2-12
- juju-cfe6b2-15
- juju-cfe6b2-14



FIGURE 4.7. VULCAN Web Application - OpenStack:Newton Assets View

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

Assets View

The selected **juju-cfe6b2-1** - hostname, has these installed packages that may be vulnerable:

- [accountsservice](#)
- [gnupg](#)
- [lvm2](#)
- [openssl](#)
- [dosfstools](#)
- [util-linux](#)
- [systemd](#)
- [lxd](#)
- [cpio](#)
- [bash](#)
- [rsync](#)
- [libxml2](#)
- [kbd](#)
- [perl](#)
- [dpkg](#)
- [python](#)
- [tar](#)
- [grep](#)
- [curl](#)
- [wget](#)
- [bzip2](#)
- [sudo](#)



FIGURE 4.8. VULCAN Web Application - OpenStack:Newton Vulnerable Installed Packages per Host

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

Assets View

The selected **openssl** - package, has been affected by these publicly known vulnerabilities:

- [CVE-2016-6302](#) has affected [cpe:/a:openssl:openssl:1.0.2g](#)



FIGURE 4.9. VULCAN Web Application - OpenStack:Newton Found Vulnerability for a Selected Installed Package

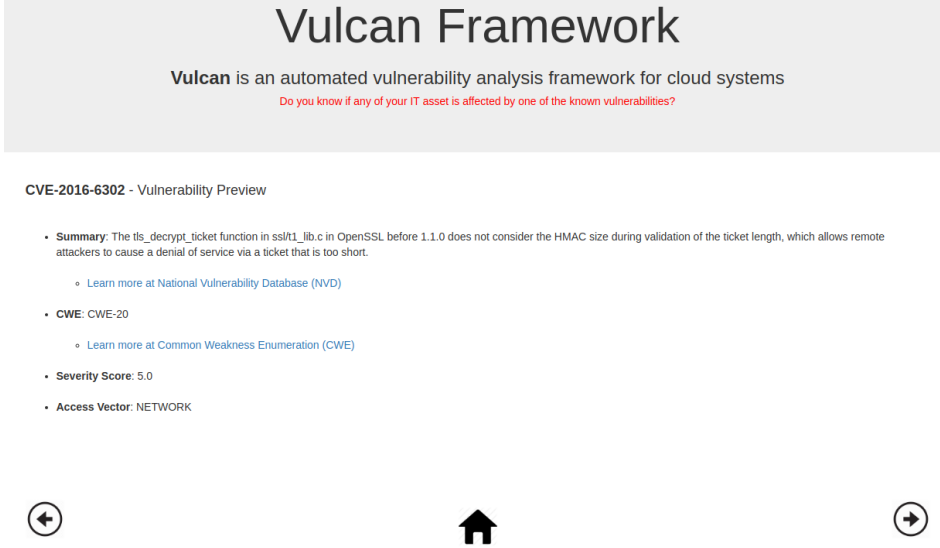


FIGURE 4.10. VULCAN Web Application - OpenStack:Newton Selected Vulnerability Identifier Details

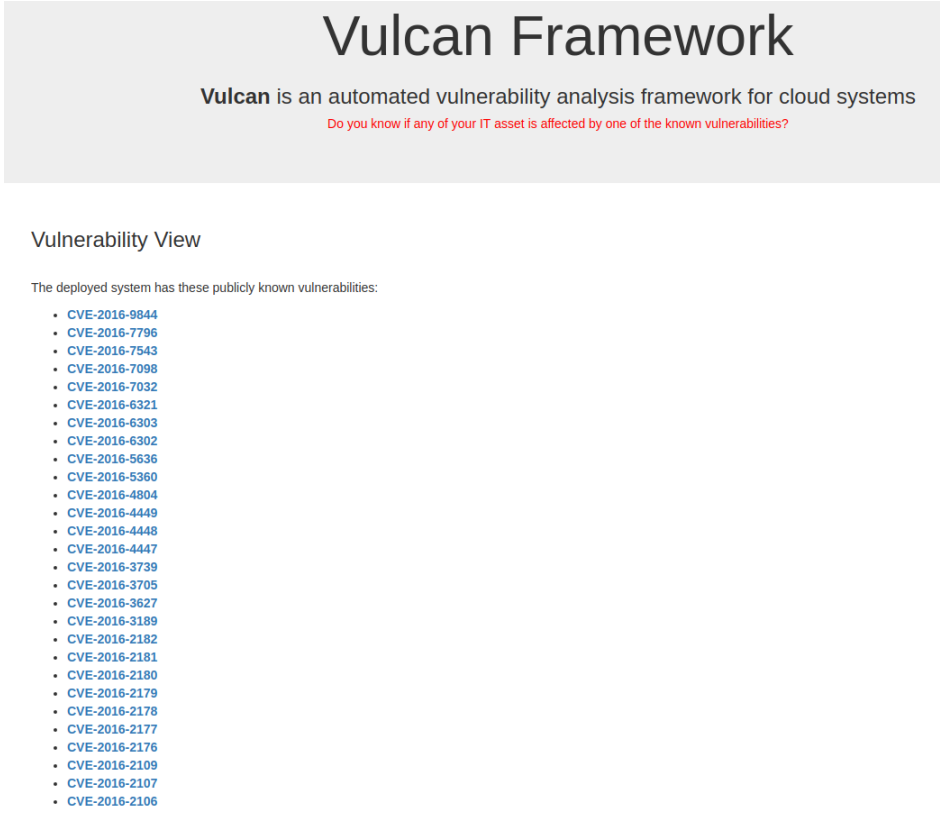


FIGURE 4.11. VULCAN Web Application - OpenStack:Newton Vulnerability View – Sample

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

CVE-2016-3189 - Vulnerability Preview

- **Summary:** Use-after-free vulnerability in bzip2recover in bzip2 1.0.6 allows remote attackers to cause a denial of service (crash) via a crafted bzip2 file, related to block ends set to before the start of the block.

- [Learn more at National Vulnerability Database \(NVD\)](#)

- **Severity Score:** 4.3

- **Access Vector:** NETWORK



FIGURE 4.12. VULCAN Web Application - OpenStack:Newton Selected Vulnerability Identifier Details

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

Exploits View

The deployed system has these publicly known exploits:

- [39918](#)
- [39887](#)
- [39768](#)



FIGURE 4.13. VULCAN Web Application - OpenStack:Newton Exploit View

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

39768 - Exploit Preview

- **Description:** OpenSSL - Padding Oracle in AES-NI CBC MAC Check
 - [Learn more at Exploit Database](#)
- **Type:** dos
- **Platform:** multiple
- **Port:** 0
- **PoC Exploit for:** [CVE-2016-2107](#) Vulnerability



FIGURE 4.14. VULCAN Web Application - OpenStack:Newton Selected Exploit Identifier Details

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

CVE-2016-2107 - Vulnerability Preview

- **Summary:** The AES-NI implementation in OpenSSL before 1.0.1t and 1.0.2 before 1.0.2h does not consider memory allocation during a certain padding check, which allows remote attackers to obtain sensitive cleartext information via a padding-oracle attack against an AES CBC session. NOTE: this vulnerability exists because of an incorrect fix for CVE-2013-0169.
 - [Learn more at National Vulnerability Database \(NVD\)](#)
- **CWE:** CWE-200
 - [Learn more at Common Weakness Enumeration \(CWE\)](#)
- **Severity Score:** 2.6
- **Access Vector:** NETWORK
- **Can be exploited by:**
 - [39768](#) Exploit



FIGURE 4.15. VULCAN Web Application - OpenStack:Newton Selected Vulnerability Identifier Details

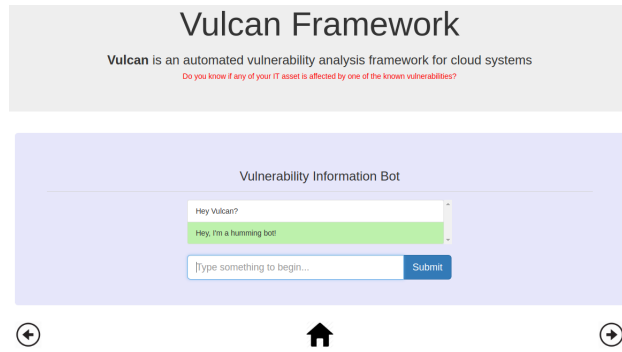


FIGURE 4.16. VULCAN Web Application - OpenStack:Newton Chatbot First User Interaction

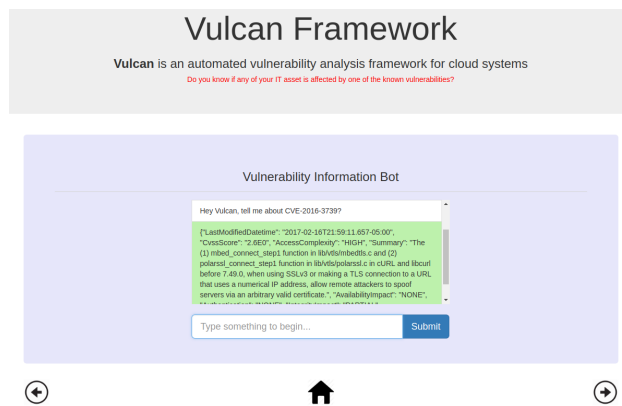


FIGURE 4.17. VULCAN Web Application - OpenStack:Newton Chatbot Sample Answer to a User Query

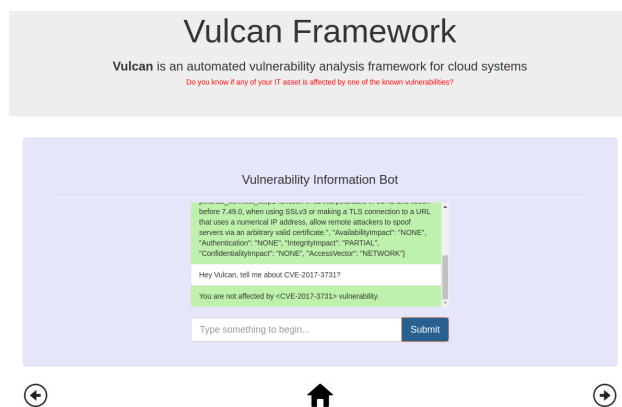


FIGURE 4.18. VULCAN Web Application - OpenStack:Newton Chatbot Sample Answer to a User Query

4.3. Ranking Cloud System Vulnerabilities[†]

In this Section, we present how to rank vulnerabilities so that security risks can be quantified. Some of this work was previously published [93] and builds upon our VULCAN foundation presented in the previous Section 4.2.

4.3.1. Overview

There have been numerous reports in the media [125], [27], [31], [57] that many financial institutions have been victims of devastating security attacks on their online services. Most of the successful attacks were based on known system vulnerabilities that were not proactively addressed. Some vulnerabilities may have appeared to pose a negligible security threat, or the organization made a trade-off in their security management.

There have been many approaches explored for addressing the challenges of managing security risks. Prioritizing management, understanding risks associated with different types of vulnerabilities, and tolerance to different types of attacks allows an organization to trade-off risks and costs [65]. Security risks need to be quantified to understand the acceptable level of risk to tolerate, or the level of trust that can be associated with the security management processes. Cloud service providers must ensure that their systems meet a specified level of security (i.e., SSLA: Security Service Level Agreement). Maintaining such SSLAs is becoming more difficult as new security attacks based on both known and yet unreported vulnerabilities continue to emerge.

We propose an approach to track and quantitatively analyze security risks associated with a system. We use the Common Vulnerability Scoring Systems (CVSS) for quantifying a vulnerability and attack graphs to assess the security risks of the target system. We extend the CVSS metrics to create an aggregate measure for systems with multiple vulnerabilities across multiple components of the system (hardware, software, and networks). The aggregate measure relies on attack graphs that show dependencies among the vulnerabilities that lead

[†]Section 4.3 is reproduced from P. Kamongi, S. Kotikela, M. Gomathisankaran, and K. Kavi, “A Methodology for Ranking Cloud System Vulnerabilities,” Proceedings of the Fourth International Conference on Computing Communication and Networking Technologies (ICCCNT) 2013, Namakkal, Tamil Nadu, India, 4-6 July 2013, with permission from IEEE.

to specific types of attacks. An organization can define different priorities for different types of attacks based on their business models.

Our model is illustrated in Fig. 4.19. The model starts with a vulnerability discovery process (powered by the VULCAN Framework shown in Fig. 4.1) for a cloud application (or a system) and produces a rank ordered list of vulnerabilities and a rank ordered list of security threat types.

Such information can be used for:

- (1) Security Auditing
- (2) Risk Assessment and Risk Management
- (3) Vulnerability Management
- (4) Cyber Security Automation
- (5) Cloud Computing Security

In the subsequent Section 4.3.2 to Section 4.3.4, we will provide the background pertinent to this work and then present our model for risk quantification.

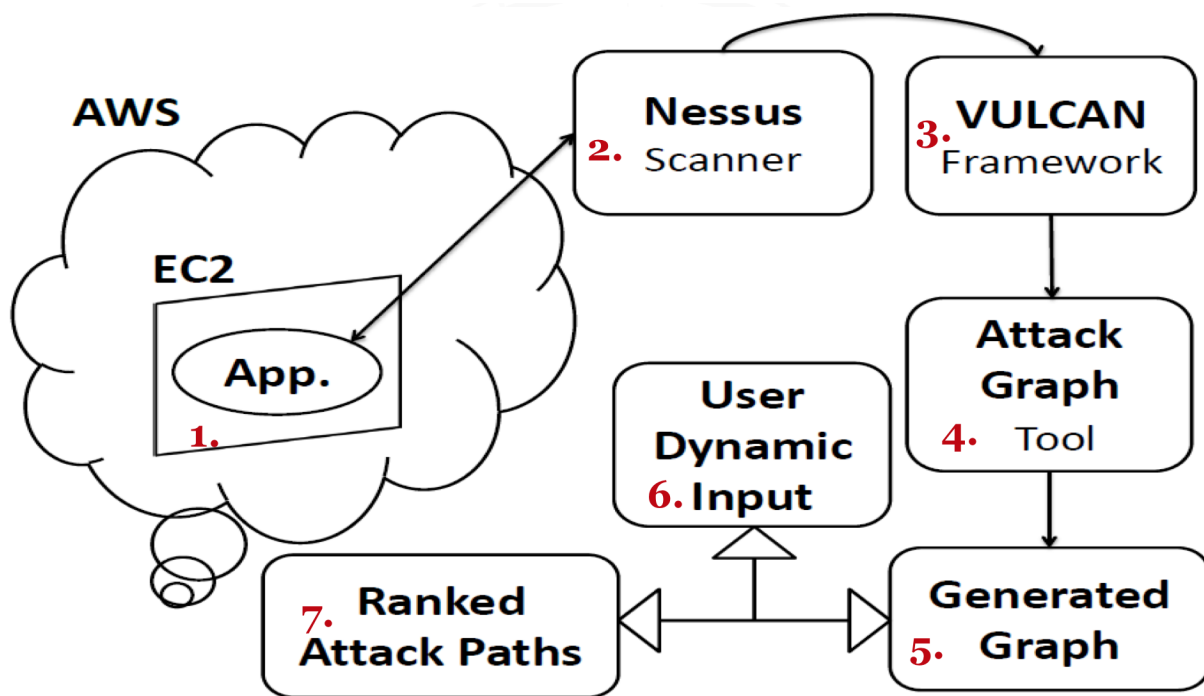


FIGURE 4.19. Vulnerabilities Ranking Methodology

4.3.2. Background

4.3.2.1. On-Fly Authentication Application

The On the-Fly Authentication Application is a prototype (shown in Fig. 4.20) hosted on Amazon Web Service (AWS) [11]. We used the Amazon Elastic Compute Cloud (EC2) [9] service, which allowed us to deploy and run an Amazon instance that supports our application. The application is made of two components, the client and server side. The client side provides a webpage form (an example is shown in Fig. 4.20) that asks the user to provide:

- (1) The direct Uniform Resource Locator (URL) of the file that needs to be authenticated (e.g., <https://www.openssl.org/source/openssl-1.0.2n.tar.gz>)
- (2) The Hash Value of the file that needs to be authenticated (e.g., 0ca2957869206de193603eca6d89f532f61680b1)
- (3) The Hash Function to be used on the file to be authenticated (e.g., SHA1)

The user then submits the form to be processed on our server side. The server end receives the user inputs and performs the following:

- (1) In a sandbox environment, the server downloads the file.
- (2) Using the provided hash function (e.g., “SHA1”), the server computes the hash value for the file.
- (3) The server compares the new hash value to the user provided hash value and returns a confirmation message if the hash values match (or denial message if they do not match). An example is shown in Fig. 4.22.

Throughout this work, we use this application to illustrate how we could use our proposed security metrics to rank assessed vulnerabilities.

4.3.2.2. Amazon Elastic Compute Cloud (EC2)

Amazon’s Elastic Compute Cloud (EC2) is a web service that provides compute capacity that can be scaled to meet user needs [9]. It is provisioned from Amazon Machine Images (AMI) [10] where an AMI is a special type of pre-configured operating system and

PINOCCHIO

Pinocchio is an On-Fly Authentication Application


Authenticate

:: To verify the integrity of any file resource on the internet, do provide its details:

URL Address:

Hash Value:

Hash Function:



[Need help?](#)

FIGURE 4.20. On-the-Fly Authentication Application – User Interface

virtual application software which is used to create a virtual machine (instance) within the EC2 [9].

Our Amazon EC2 instance includes a Linux kernel, AWS tools and repository access to multiple versions of IT products such as MySQL, PostgreSQL, Python, Ruby, and Tomcat. On top of these, we installed the Apache HTTP Server (“httpd”) for running any web services on the EC2 instance. The web server is reachable on ports 80 (HTTP) and 443 (HTTPS) over TCP. We allow SSH access (TCP port 22) from any source. These rules are defined as part of the security groups [8] setting (firewall configuration).

4.3.3. Our Approach for Security Risk Quantification

4.3.3.1. Attack Graph Generation

- (1) **Specifications:** In our approach to producing security metrics that assess our cloud application introduced in Section 4.3.2.1, we need to understand the system



FIGURE 4.21. On-the-Fly Authentication Application – Example

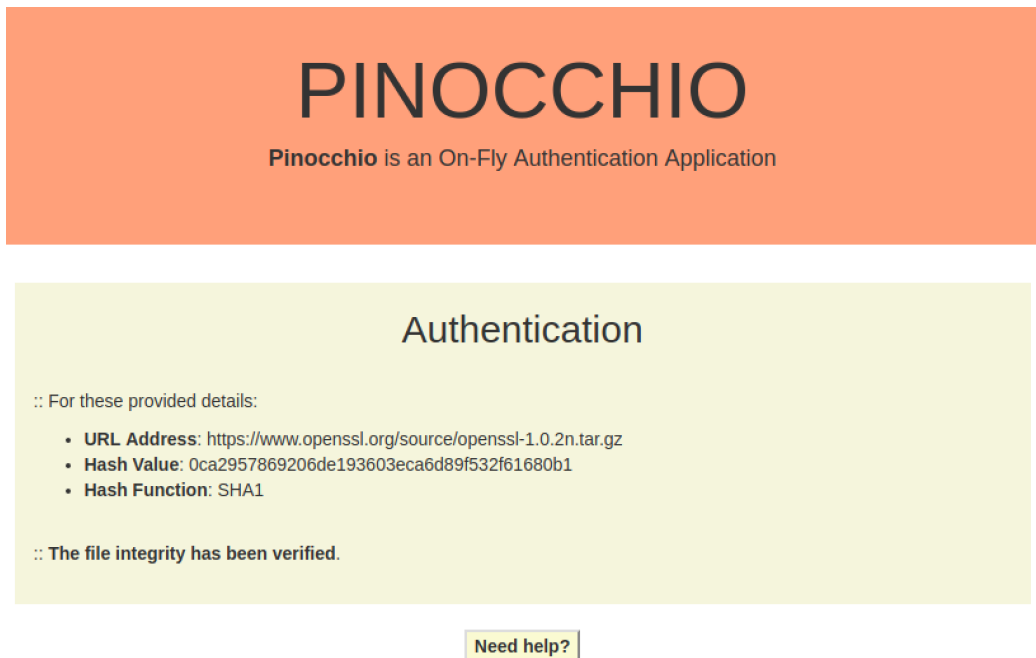


FIGURE 4.22. On-the-Fly Authentication Application – Example Evaluation

as viewed by an attacker or by the internal organization. We can use tools such as security scanners to determine what an attacker can see. We use the Nessus [105]

scanner to obtain our target application network description and any available vulnerability information. In addition, we add the security vulnerability data obtained from our VULCAN tool described previously to enhance the knowledge for assessing security risks. Our model can be more accurate if additional and specific knowledge about the system being assessed is provided (for example how components are configured, if patches are in place, and other information can be collected by tools such as LEGOS described in Chapter 3). To illustrate our model, we currently assume a simple network description with vulnerability information acquired by the Nessus scanner.

- (2) **Modeling:** The information collected allows us to create a baseline for our attack graph generation. We utilize one of the currently adopted approaches such as the Buchi automaton and SPIN model checker to verify if the information acquired thus far are correct and complete; otherwise, correct the information before generating attack graphs [109].
- (3) **Generation:** Once the baseline for the attack graph is collected, we use our scripts to generate attack graphs. We can use any known methods for generating attack graphs (for example the works presented in [109] or [99]). We can use [99] for generating attack graphs and add the Common Vulnerability Scoring System (CVSS) metrics with the graphs to obtain risk scores associated with the system.
- (4) **Analysis:** The generated attack graph defines all possible paths that an attacker can take to achieve specific goals (for example, obtain authorized access to a system). This approach is straightforward to assess when faced with a couple of possible attack paths. What if the attack graph contains a multitude of possible attack paths? To answer this problem, we explored all currently known analysis techniques such as the use of a Probability Scenario Graph (PSG) as described in Oleg’s work [109]. Also, some other related techniques have been devised to reduce the number of attack paths to a number of more likely ones that cause an imminent threat. Our contribution in this work is based on two algorithms crafted to use the concepts of

security metrics to determine the system or application tolerance level in a quantified representation. We also provide security analysts with the means to prioritize attack paths using our ranking methodology.

4.3.3.2. Common Vulnerability Scoring System (CVSS)

CVSS is a vulnerability scoring system designed to provide an open and standardized method for rating IT vulnerabilities [36]. CVSS helps organizations prioritize responses to security vulnerabilities by communicating the base, temporal and environmental properties of a vulnerability.

We add a Weighted Average Mean metric to the CVSS, which allows us to rank attack paths generated as described in the previous section (Section 4.3.3.1). Our approach contains two parts. The Ranking Algorithm implements the first part (See Algorithm 2) and the second part is a graphical user interface (labeled Dynamic Input Specification, See Algorithm 3) that receives weights that define the significance of different attack paths as perceived by the user (Algorithm 3 feeds weights to Algorithm 2).

Below are some examples illustrating the dynamic input formats that are supported:

- First example is an attack vector, e.g., “AV:L/AC:M/Au:N/C:N/I:P/A:C”. In this example, a vulnerability has base metric values of “Access Vector:Low, Access Complexity:Medium, Authentication:None, Confidentiality Impact:None, Integrity Impact:Partial, and Availability Impact: Complete”.
- Second example is the Weighted Average Mean input, e.g., “BM:0.25/TM:0.25/EM:0.5”. In this case, a vulnerability has assigned Weights for groups of “Base Metric:0.25, Temporal Metric:0.25, and Environmental Metric:0.5”. Here we observe that the sum of the weights equals one.

4.3.4. Example Workflow

An overview of how our proposed approach works is illustrated in Fig. 4.19.

- We have developed a web application on Amazon EC2 (presented in Section 4.3.2.1).

Algorithm 2: Ranking Using CVSS metrics

Require: An attack graph G with CVE-ID assigned on each node

for *The given attack graph G* **do**

 Create a list L of all attack paths from G

 Display a menu of CVSS metric classes along with their subclasses

 Prompt the user for dynamic inputs (DI)

for *Each attack path in L* **do**

 Extract CVE-IDs from the attack path

 Map CVE-IDs to their reported CVSS metrics that matches the user DI

 Initialize a custom data type L' with the current attack path entry details

 Calculate the overall Weighted Average Mean and add it to L'

end

 Sort L'

end

Return top-ranked attack paths from L'

Ensure: A ranked list of attack paths based on the user dynamic inputs

- Using the application IP address, we gather information using the Nessus [105] scanning tool.
- We have conducted Web App Tests and an External Network Scan on our application.
- Web App Tests are used to discover known and unknown vulnerabilities in web applications.
- External Network Scan is used to scan externally facing hosts.
- In the External Network Scan, all 65535 ports and known web application vulnerabilities are exercised.
- From the information collected, we identify specific vulnerabilities and add additional vulnerability information using VULCAN, which uses the system specifications obtained from the scanner.
- The specifications plus the vulnerability information collected above are fed to the

Algorithm 3: User Dynamic Input Specifications

Require: Metrics

- CVSS Metric Groups
 - (1) Base Metric (BM)
 - Access Vector (AV)
 - Access Complexity (AC)
 - Authentication (AU)
 - Confidentiality Impact (C)
 - Integrity Impact (I)
 - Availability Impact (A)
 - (2) Temporal Metric (TM)
 - Exploitability (E)
 - Remediation Level (RL)
 - Report Confidence (RC)
 - (3) Environmental Metric (EM)
 - Collateral Damage Potential (CDP)
 - Target Distribution (TD)
 - Security Requirements (CR, IR, AR)
- Weighted Average Mean (0 - 1)
 - Default values:
 - * Low (0.25)
 - * Medium (0.5)
 - * High (0.75)
 - * Not defined (0)
 - * Top priority (1)

attack graph generation tool to create all possible attack paths that may be used by an attacker.

- Output from the above step is a complete attack graph. Using our algorithms presented in Section 4.3.3.2, we then extrapolate all possible attack paths.
- Additional inputs are then requested from the user to prioritize and generate attack paths using our ranking algorithm 2.

4.3.5. Summary

In [93], we extended our previous work on VULCAN [92], a Vulnerability Assessment Framework for Cloud Computing. In VULCAN, we modeled security vulnerabilities and defined a vulnerability ontology that classifies them. We then developed an automated process to instantiate our ontology using the data provided by NVD [85] which resulted into our ontology knowledge base (OKB). Using this OKB, we can study and assess security vulnerabilities of individual components or subsystems of a cloud computing system. We achieve this complete assessment via VULCAN components, such as Semantic Natural Language Process (SNLP), and modules like System Classifiers and Indexer (see Figure 4.1 for illustration.)

In addition, we have a method for ranking vulnerabilities. We conceptualize cloud system vulnerabilities as nodes in an attack path that can be used by an attacker. We generate an attack graph to represent all possible attack paths. We can then assign a risk score using the CVSS metric for each vulnerability, and probability models to aggregate the overall risk associated with the attack graph.

4.4. Prediction Model for Unknown Vulnerabilities[‡]

In this Section 4.4, we present our approach to predicting yet undiscovered vulnerabilities using software metrics and software maturity models. Some of these results were previously published in [90]. This work builds on our VULCAN framework that was presented in Section 4.2.

[‡]Section 4.4 is reproduced from P. Kamongi, K. Kavi, and M. Gomathisankaran, “Predicting Unknown Vulnerabilities using Software Metrics and Maturity Models,” Proceedings of the Eleventh International Conference on Software Engineering Advances (ICSEA) 2016, Rome, Italy, August 21-25, 2016, with permission from the International Academy, Research and Industry Association (IARIA).

4.4.1. Overview

Any software product that is in production goes through a series of changes throughout its lifecycle as a result of feature changes or bug fixes among other factors. As a software product matures, it has been shown that it is vulnerability-prone and that its security vulnerabilities do get discovered throughout its maturation. For the last decade or so, we have seen a trend of security vulnerabilities in software products being disclosed on a regular basis [85]. This observed trend calls for increased security awareness and demands new approaches to stay ahead of security threats from yet unknown or disclosed vulnerabilities.

Security vulnerabilities that are discovered and leveraged by malicious actors before the software provider becomes aware of them and fixes them are known as zero-day (0day) vulnerabilities. The worrisome nature of 0day vulnerabilities is due to the endless number of approaches that a malicious actor might employ to attack a software product. The security community and software product vendors sponsor bug bounty initiatives in an attempt to stay ahead of such zero-day attacks, but discovering software bugs or vulnerabilities before a malicious attacker discovers them is a challenge. This requires assessing the software using tools that analyze the design, implementation, and usage. Some examples include static code analysis and dynamic analysis of binaries. However, these techniques may not discover all software bugs and vulnerabilities. Our approach relies on historical approaches to predict hidden or latent software bugs based on the complexity of software. In addition, we rely on measuring how a software product matures with new releases that are supposed to fix detected bugs or vulnerabilities. However, the fixes themselves may introduce new bugs or vulnerabilities. The measure of software maturity captures these aspects.

An estimation of the potential number of undetected or unreported security vulnerabilities is useful because it may lead to proactive strategies for protecting IT assets. In this work, we want to address the following types of questions:

- To what extent do software complexity metrics correlate with the number of vulnerabilities disclosed for a software product?
- Can a set of software metrics (including complexity metrics) predict the number of

vulnerabilities contained in a software product?

Previous research has attempted to predict software error (or bug) incidences using software change history [46]. Software metrics have also been used to predict vulnerability prone codes in software [18]. Other techniques have been used to study the trend of vulnerabilities in software products [16, 126]. In this study, we explore the correlation between software change history and maturity with the number of vulnerabilities each software release may contain and subsequently discovered.

For this purpose, we created the following:

- **SWEEP**: a toolkit that automates software complexity metrics generation and analysis (our prototype web application is illustrated in Fig. 6.20).
- A methodology for using the SWEEP toolkit to generate a dataset for a software product automatically. The dataset produced contains information on all releases of a software product along with the relevant software metrics (metrics that have been known to correlate with software bugs and vulnerabilities) and the number of reported vulnerabilities for each release in a timeline fashion (forming the basis for tracking the maturity of the product with new releases).
- A web service that uses a machine learning regression analysis to correlate the data set collected by the SWEEP toolkit with the number of vulnerabilities associated with the specific release of the software (our PREDICTION prototype web application is illustrated in Fig. 6.21).

4.4.2. Our Predictive Methodology

In this section, we present our model for predicting the number of vulnerabilities in an IT Product.

4.4.2.1. Data Collections

For this study, we have devised a generic and automated approach for collecting required data on a software product using the source code for the product across all known versions. The specific data collected is related to well-known software metrics that have

been used to measure the complexity of a software product that are known to correlate with software bugs. In addition, we collect the number of security vulnerabilities reported with each release of the software.

For the IT software product under consideration, we start by collecting details about the product’s releases (with an identifying name), the number of releases and the number of vulnerabilities already reported. We analyze the source code of each released version of the product and collect various software metrics for source codes of each version. The collected data is stored in a dataset file as comma-separated values (CSV). The following describes the process of collecting data.

- (1) Select the IT Product to analyze.
- (2) Download all releases of this IT product’s source code for each version.
- (3) For each release, represent it using a Common Platform Enumeration (CPE) [83] format as its unique identifier for cataloging the product version and its data into our datasets.
- (4) For each release, discover all reported vulnerabilities using VULCAN indexing schemes (See. Section 4.2.1.5) or IKAWAFARM API (See. Section 3.4.1).
- (5) Analyze the source code for each version and obtain software complexity metrics using available tools such as Understand [64].

Our SWEEP toolkit automates all the steps described.

4.4.2.2. Predictive Model

The research question we originally proposed was whether we could predict the number of undisclosed vulnerabilities for any given software product. We base our solution on the data that can be collected for this software product using our SWEEP toolkit as discussed in the previous Section 4.4.2.1. Once a dataset is generated for the software product of interest, we use a machine learning regression classifier to build a model that can predict the number of vulnerabilities contained in a specific release of the software under evaluation.

The accuracy of prediction depends on the amount of data, since the more data we

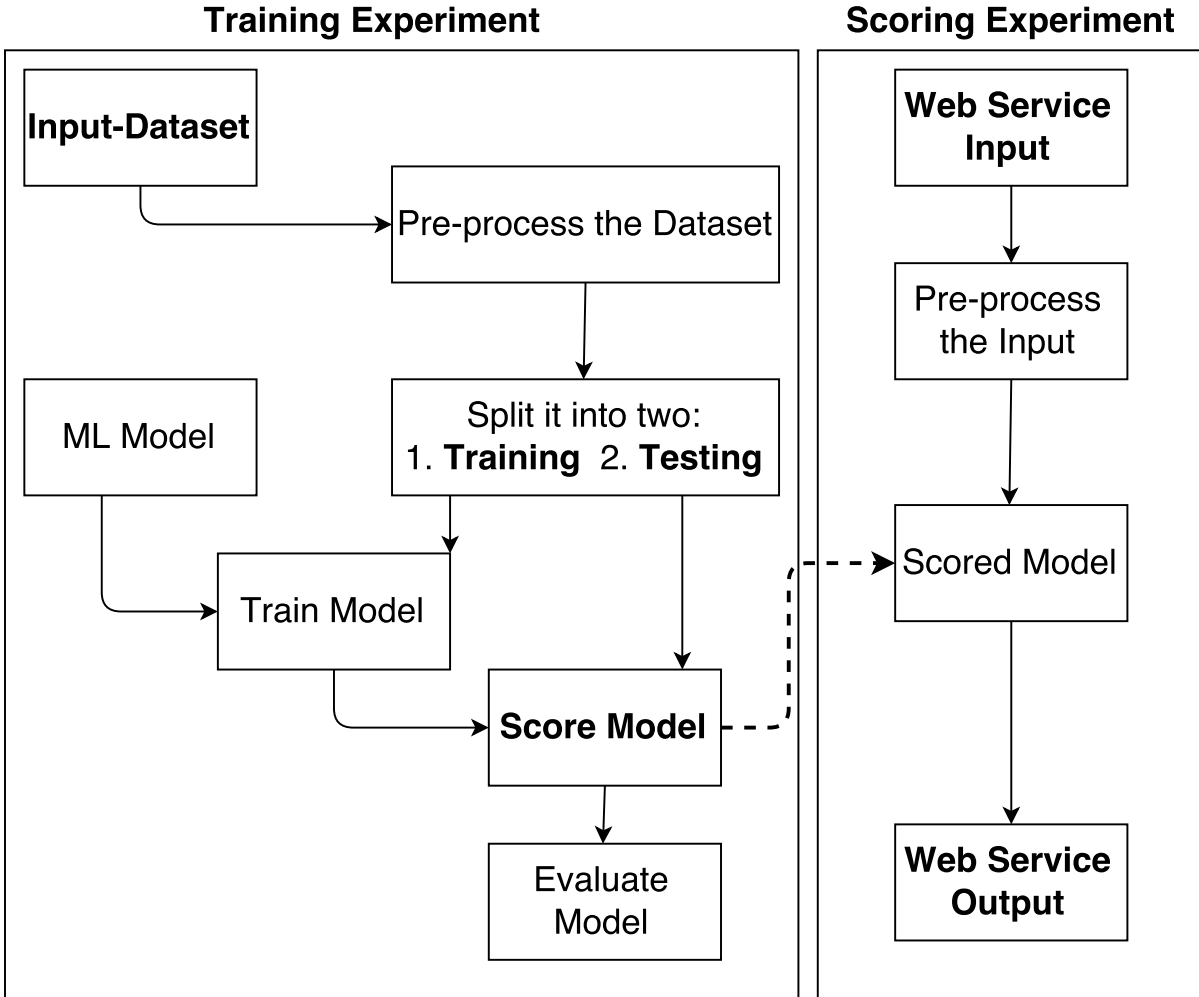


FIGURE 4.23. Predictive Model - Framework

can use during the training of the machine learning model, the more accurate the prediction will be. In our testing environment, we collected 124 software metrics on each version of the software. The specific metrics are those that have been shown to correlate well with vulnerabilities associated with the software. We used different regression techniques to find one that best fits our needs. We chose Microsoft Azure – Machine Learning Studio [13] for this purpose. Azure Machine Learning Studio provides many easy-to-use building blocks for developing a predictive solution.

Some specific aspects that drive our predictive model are our choices for the regression classifier module and feature scoring method. In Section 4.4.3, we provide details on these

choices and how they strengthen the prediction for a given IT product.

4.4.2.3. Prediction Workflow

Our approach to predicting the number of unknown vulnerabilities for any given software product follow this workflow:

- Start by generating a dataset for the given IT Product as illustrated in Section 4.4.2.1.
- Using the above dataset, build and test a predictive experiment within Azure ML Studio as illustrated in Section 4.4.2.2. Once the above predictive experiment has completed successfully, a trained model and web service will be produced.
- Using a subset of the dataset that was reserved for validation, ensure that the trained model is scoring well against this validation data.
- Expose a middleware application which leverages the above predictive web service to receive input data as illustrated in Fig. 4.23. Return the predicted number of vulnerabilities associated with a software release along with other associated metadata such as the prediction accuracy and error rates.
- We can now perform dynamic prediction for any of this IT Product's releases by passing the release version source code details for data collection. Using the generated data as input to the above middleware application, we get the predicted number of vulnerabilities for the given software product release.

The predicted number can then be interpreted with two views: one for the overall accumulated number of vulnerabilities and the other view for the unknown number of vulnerabilities (which can be easily computed by subtracting the known vulnerabilities from the predicted ones and taking into consideration the prediction error rate). Since predicted vulnerabilities cannot be classified based on the potential types of threats resulting from their exploitation, we separate them from known vulnerabilities which can be classified based on the threat types they expose. This allows us to show a separate security threat risk score for known and hidden vulnerabilities.

4.4.3. Experimental Study and Evaluation

In this section, we show how we validated our risk scoring model using an open source software OpenSSL [40].

4.4.3.1. Use Case: OpenSSL – Software Product

The discovery of the **OpenSSL**'s Heartbleed bug [84] in 2014 revealed that this vulnerability remained unknown for a couple of years before it was discovered. Heartbleed and other similar types of vulnerabilities serve as the motivation for our research concerning the prediction of hidden vulnerabilities.

To validate our model, we examined different open source software products but selected **OpenSSL** [40] because of the critical role it plays in Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. Besides, many other software products rely on OpenSSL (since it is commercial grade and open source) to build new software and services. Thus vulnerabilities, discovered and hidden, will impact many software systems and services. One additional reason is the existence of several versions and releases of OpenSSL, providing more data and longer history to train our prediction models (particularly our maturity model). However, we believe that our model can be used to predict the number of vulnerabilities associated with any new release of a software product, provided source code for all releases of the software is available, as well as information about vulnerabilities with each release.

4.4.3.2. OpenSSL – Dataset Generation

We downloaded and analyzed 154 released OpenSSL versions, with versions 0.9.x, 1.0.0, 1.0.1, 1.0.2, 1.1.0, and fips [40]. We use a directory/path to all OpenSSL versions' source code (uncompressed/compressed) as input to our SWEEP toolkit. SWEEP will analyze the source code of each version and collect software metrics as detailed in the previous section of this chapter.

This OpenSSL dataset includes 154 metrics for each OpenSSL version. Some of the metrics included are described below.

- *CPE-Name* representing each analyzed OpenSSL release (e.g., *cpe:/a:openssl:openssl:1.0.0f*).
- *Year-xxxx* representing the number of disclosed vulnerabilities in the year xxxx (where xxxx ranges from 1999, the first year when NVD [85] shows vulnerability data, until the current year).
- *#CVEIDs* representing the number of known vulnerabilities (e.g., 50 is the total number of disclosed vulnerabilities for the above CPE instance example).
- *Understand-Metric-1, ..., Understand-Metric-n* representing all computed software metrics measured by the Understand tool [64] (e.g., the above CPE instance has a *SumCyclomatic* value of 43479, and other metrics data are provided accordingly).

4.4.3.3. OpenSSL – Predictive Experiment

Consider tracking product releases (minor and major) and vulnerabilities detected for the OpenSSL product shown in Fig. 4.24. One of the observed trends in terms of the number of vulnerabilities detected across these releases can be seen in Fig. 4.24, Fig. 4.25. They show a decreasing number of vulnerabilities with major new releases, indicating software maturity with fewer bugs and vulnerabilities.

We predict a similar behavior with other IT products since this trend reflects a behavior similar to software maturity or improved quality over time. Here we see the maturity of OpenSSL in terms of the number of reported vulnerabilities over time. We also make the following preliminary observations from the data collected so far, along with this small sample illustrated in Fig. 4.24 and 4.25.

- The history of reported vulnerabilities has shown a decreasing trend throughout each IT Product’s minor releases (e.g., OpenSSL:1.0.1.:a, b, c, ..., j) in terms of the number of vulnerabilities, with a few exceptions for some limited distribution versions (see Fig. 4.25).
- The average number of reported vulnerabilities spiked with a major new release of OpenSSL following a large number of minor releases (this can be seen in Fig. 4.24).
- The newly discovered vulnerability in a current IT product release is likely contained in previous releases because the releases share some common technologies

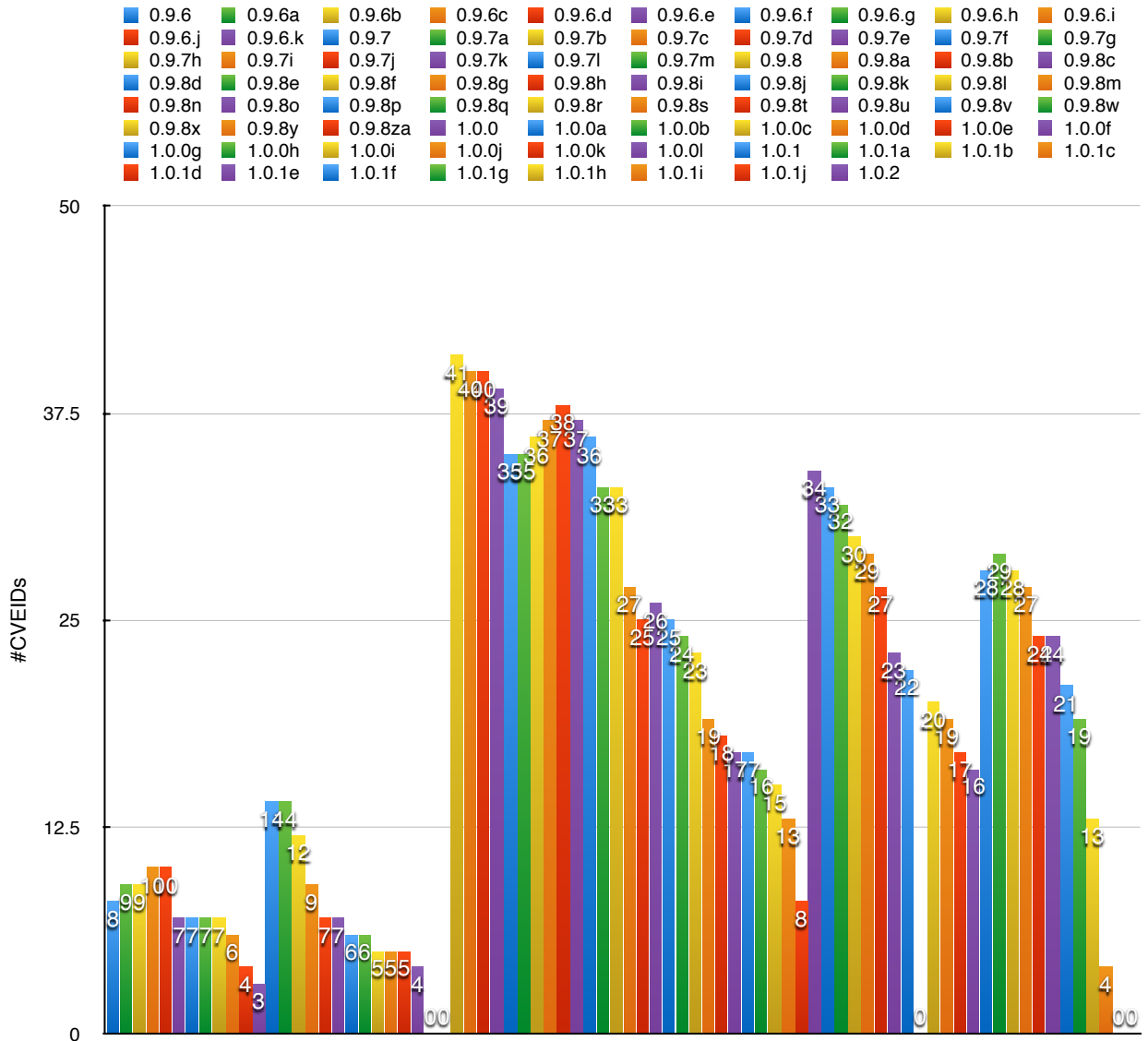


FIGURE 4.24. OpenSSL Releases vs. Disclosed Number of Vulnerabilities

(and common vulnerabilities).

- The straight line in Fig. 4.25 reflects the mean root square (R^2) value (coefficient of determination) which appears to be above 0.5 when using Linear, Polynomial and Exponential trend estimations. However, while using logarithmic and power trends, the R^2 value was less than 0.5. This hints that there is value in knowing how the data can be used to predict the number of vulnerabilities, although we need a more carefully designed approach to correlate the data with vulnerabilities.

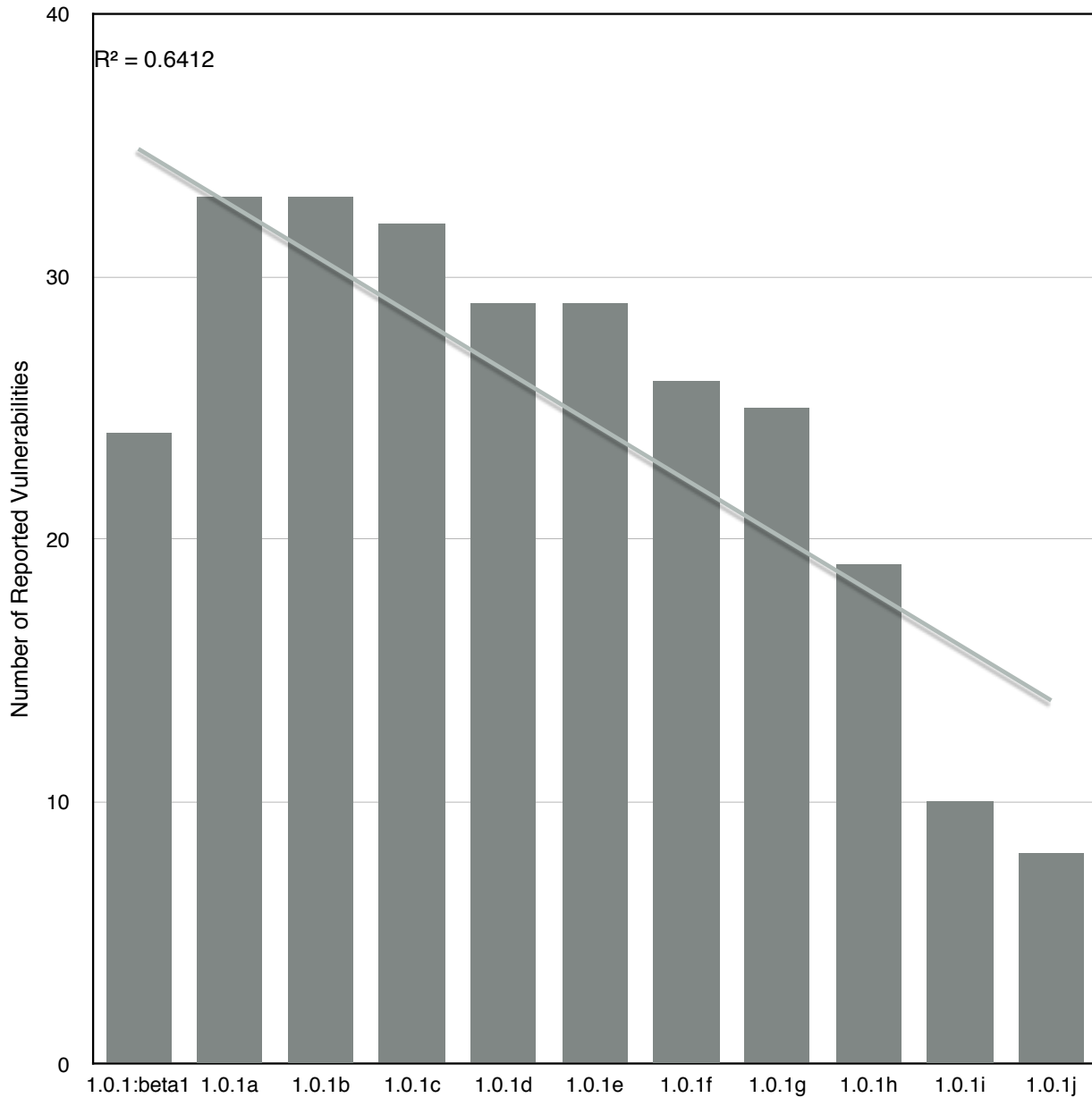


FIGURE 4.25. OpenSSL:1.0.1 – Minor Releases Linear Trend

To improve our model and produce a higher coefficient of determination with the prediction of the number of vulnerabilities, we considered all the features of the vulnerability details and generated source code metrics to build a predictive model using machine learning techniques as presented in previous sections. Using Azure ML Studio [13], we built our predictive experiment as illustrated in Fig. 4.23.

The first step is understanding the nature of the training dataset which requires pre-

processing the data to remove noise or outliers and selecting the most significant features that lead to highly confident predictions. Azure ML Studio offers the necessary tools to facilitate these pre-processing tasks such as Project Columns and Filter-Based Feature Selection. For the Filter Based Feature Selection module, we used the Pearson correlation feature scoring method for identifying the most significant features from the data sets. For our data sets, this process selected 25 metrics. The next step is deciding how many data sets to use for training the machine learning model, and how many data sets to use for validating the trained model. In Azure, this is achieved by using the Split module. We chose a 50-50 split for training and validating the model.

We found that regression-based models are well suited for our purpose. We explored all available Azure ML Studio's [13] machine learning regression models and determined that a *Boosted Decision Tree Regression* [61] is the best for our predictions. By following the easy-to-use predictive framework illustrated in Fig. 4.23, we trained and scored our chosen model using the generated OpenSSL dataset (from the previous section) targeting the `#CVEIDs` feature name (as presented in Section 4.4.3.2). Table 4.2 includes the results obtained from our model. We present a detailed discussion of our results in Section 4.4.3.6.

Table 4.3 shows a comparison of the number of reported vulnerabilities and the number of predicted vulnerabilities for some OpenSSL releases used for the training experiment (these results validate the model using training data sets). The difference between predicted and known vulnerabilities can be explained in part using the Mean Absolute Error with the prediction as shown in Table 4.2. In addition, it is not clear that the particular release has all vulnerabilities reported or if additional vulnerabilities could be detected in the future (if the release is still used in the public domain). To be on the conservative side we use the Mean Absolute Error to predict an upper bound on the total number of vulnerabilities contained in a given software product release.

4.4.3.4. OpenSSL – Predictive Model Validation

In Table 4.4, we present a comparison of the OpenSSL releases used to evaluate and validate the scored predictive model. These results tell us how well our model is able to

Number of Desired Features	Mean Absolute Error	Relative Absolute Error	Coefficient of Determination
5	7.995	0.618	0.597
10	3.341	0.258	0.927
15	3.347	0.258	0.926
25	3.780	0.292	0.917
40	3.985	0.308	0.912
65	3.851	0.297	0.916
80	3.597	0.278	0.918
95	3.597	0.278	0.918

TABLE 4.2. Using our Labeled OpenSSL Dataset to Evaluate its Boosted Decision Tree Regression Model

OpenSSL Releases	Known #Vulnerabilities	Predicted #Vulnerabilities
cpe:/a:openssl:openssl:0.9.8s	22	17.139
cpe:/a:openssl:openssl:1.0.1f	53	41.747
cpe:/a:openssl:openssl:0.9.7g	35	30.823
cpe:/a:openssl:openssl:1.0.1g	52	43.777
cpe:/a:openssl:openssl:0.9.6m	30	37.448

TABLE 4.3. Sample of Scored OpenSSL Instances

score against our validation dataset.

- Mean Absolute Error : 2.927
- Root Mean Squared Error : 4.068
- Relative Absolute Error : 0.203
- Relative Squared Error :0.059

- Coefficient of Determination :0.940

OpenSSL Releases	Known #Vulnerabilities	Predicted #Vulnerabilities
cpe:/a:openssl:openssl:1.0.1a	60	52.498
cpe:/a:openssl:openssl:1.0.0h	46	46.815
cpe:/a:openssl:openssl:0.9.8m	34	33.867
cpe:/a:openssl:openssl:1.0.2c	15	15.729
cpe:/a:openssl:openssl:1.0.2d	14	11.526

TABLE 4.4. Sample of OpenSSL Instances for Validation

4.4.3.5. OpenSSL – Prediction of Unknown Vulnerabilities

Transforming the previously built predictive model for OpenSSL into a web service endpoint is straightforward. In Table 4.5, we show some examples of the predicted number of vulnerabilities for some versions of OpenSSL releases that are currently used. These OpenSSL instances have no reported vulnerabilities. Therefore, the predicted number reflects the unknown number of vulnerabilities that may be discovered in the future.

4.4.3.6. Discussions

In Table 4.2, we presented results from our machine learning model (Boosted Decision Tree Regression) for predicting vulnerabilities in OpenSSL software product releases. The following preliminary observations can be made from these results.

- The *OpenSSL* dataset fits well with the selected machine learning technique, yielding a high coefficient of determination above 0.5 (which is better than a random guess).
- The *OpenSSL* predictive model scoring shows a positive correlation between the known vulnerabilities (#CVEIDs) and predicted ones (Scored Labels), which can be viewed via the scatter plot generated within the Azure ML Studio workspace. This validates our hypotheses that we can predict the number of vulnerabilities contained in an IT Product using software metrics and vulnerability disclosure history.

OpenSSL Instances	Known #Vulnerabilities	Predicted #Vulnerabilities
openssl-1.0.0:beta1	0	11.054
openssl-1.0.0:beta2	0	11.054
openssl-1.0.2:beta1	0	11.667
openssl-1.0.2:beta2	0	11.667
openssl-1.1.0:pre2	0	8.641
openssl-engine:0.9.6m	0	11.148
openssl-fips:2.0.9	0	11.148
openssl:0.9.8zd	0	5.489
openssl:1.0.0t	0	8.843

TABLE 4.5. OpenSSL’s Versions – Predicted Number of Vulnerabilities

- The coefficient of determination of the model we scored is at the lowest when the desired number of features is set to 5. By taking a close look at the automatically selected five features (Year-2010 to Year-2014), it reveals that these features are primarily related to the vulnerability discloser history, and thus based on software maturity.
- The coefficient of determination is improved and reaches its highest when the desired number of features is increased from 5 to 10 or higher value. This improvement in the prediction accuracy is a result of our feature selector capabilities used to identify additional features that measure software complexity based on the source code (such as *CountLineCodeExe*, *Knots*, *CountPath*, *SumEssential*, *Cyclomatic number*).

Table 4.5 contains results of our evaluation of the OpenSSL dataset for new or beta instances (or releases) of the product with no reported vulnerabilities thus far. It should be noted that these two OpenSSL instances (*openssl-1.0.2:beta1* and *openssl-1.0.2:beta2*) have very similar source code bases. Therefore we predict that both versions will likely contain the same number of vulnerabilities. These vulnerabilities should be viewed as the

potential number of vulnerabilities that will probably be discovered in these products. This information can be used to plan for defensive mechanisms to mitigate security risks from the unknown vulnerabilities.

4.4.3.7. Recommended Proactive Strategies

Ideally one should be able to implement countermeasures to patch or mitigate risks from known vulnerabilities. Many IT organizations plan for such mechanisms [56, 55]. However, the rate at which new vulnerabilities are being detected and reported is making it difficult to maintain up-to-date lists of patches and other countermeasures. Moreover, as we have shown in this work, IT products very likely contain unknown or yet to be discovered vulnerabilities. Thus, it is necessary to explore additional (beyond patching) defensive measures to increase our confidence in IT products. We include some recommendations in this regard.

- Any unknown pattern or behavior observed for an IT Product being assessed via security penetration testing approaches or monitored via deployed security infrastructures can serve as an indicator that zero-day (or undiscovered) vulnerabilities may be present or being exploited in the IT product of interest. The predicted number of vulnerabilities should be used as a guideline for maintaining vigilance and for keeping the process of penetration testing and observing any abnormal behaviors.
- We also recommend exploring various software rejuvenation techniques in an attempt to mitigate some malware that may be exploiting hidden or unknown vulnerabilities before taking a foothold in the product. It has been shown that software rejuvenation [67] can minimize security risks due to malware. It has also been shown how the cost of rejuvenation can be used to plan the frequency of rejuvenation schedules.

4.4.4. Summary

We have presented our novel approach for predicting the number of unknown vulnerabilities in a given IT Product. We have shown how to generate a dataset that represents

product maturity in terms of source-code base growth and vulnerability disclosure history. We have shown how to use such a dataset and develop a ML model that results in accurate predictions. We used the Azure cloud-based machine learning framework for this purpose. We validated our approach using the OpenSSL IT product. We plan to test this approach with other open source products with a long history of new releases.

Our approach for analyzing the source code of a given IT Product and leveraging its vulnerability disclosure history toward building a predictive model serves as a basis for building other solutions. A possible extension to our model is to categorize the predicted number of vulnerabilities into threat types (i.e., STRIDE [62]) using some inherent IT product properties along with some actionable threat intelligence and, in turn, propose relevant mitigation techniques to counter these vulnerabilities and threats.

The other aspect of our work that we plan to extend is the ability to design and train our predictive model in a generic way that would allow IT products that may need different machine learning and training approaches. We plan to group IT products into categories (for example, based on the functionalities provided by them), identify representative features of products that belong to a category and design an approach for predicting vulnerabilities across a broad set of products. Finally, we plan to expand on the IT product features to enhance our prediction accuracies using security threat intelligence reports, inherent vulnerabilities associated with different programming languages and development platforms.

In the following Chapter 5, we present the NEMESIS architecture which leverages all the work presented thus far, along with novel techniques for threat modeling and risk assessment for cloud systems.

CHAPTER 5

ONTOLOGICAL BASED THREAT MANAGEMENT*

In this Chapter, we present an automated threat modeling and risk assessment approach. The description here is based on our previously published work [91]. This work leverages the techniques presented in Chapters 3 & 4.

5.1. Overview

Cloud computing is defined as the delivery of on-demand computing resources, everything from applications to data centers over the Internet on a pay-for-use basis [19]. The design principle revolves around a custom or an open source cloud operating system that is in charge to control and provision requested resources throughout the data center(s). Cloud computing services can be classified as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). For example, the OpenStack [95] cloud operating system enables developers to design a cloud computing deployment either as a Public, Private or Hybrid cloud and support computing services. It is critical to ensure that cloud computing services can be protected against security threats.

To protect against attacks, the administrator must perform a regular security assessment of the systems by looking for known vulnerabilities and known attacks. This task can be very complex both because cloud computing systems are complex and also because they rely on many shared technologies and third-party software products. According to Cloud Security Alliance (CSA) Cloud Computing Top Threats in a 2013 report [7], “shared technology vulnerabilities” was ranked among the top threats facing cloud computing ecosystems. Shared technology vulnerabilities have implications since a compromise of any of these technologies such as a hypervisor, a shared platform component, or an application, comprises all customers that share the technology [7]. It has been shown that cloud computing threats

*Chapter 5 is reproduced from P. Kamongi, M. Gomathisankaran, and K. Kavi, “Nemesis: Automated Architecture for Threat Modeling and Risk Assessment for Cloud Computing,” In: ASE Big-Data/SocialInformatics/PASSAT/BioMedCom 2014 Conference, ISBN: 978-1-62561-003-4, Harvard University, December 14-16, 2014, with permission from the Academy of Science and Engineering (ASE).

are influenced by different agents, where some are the result of inherent vulnerabilities found in shared technologies used, and others come to life due to the composition of services of shared technologies.

Confidentiality, integrity, availability, consistency, control, and audit are the most critical aspects of cloud assets that must be protected [39]. When assessing security threats, it is useful to ask questions like:

- How can an attacker change the authentication data?
- What is the impact if an attacker can read the user profile data?
- What happens if access is denied to the user profile database?

Using this analogy for thinking about threats, there exist threat models like STRIDE [62] that help us understand the types of attacks and consequences of such attacks. As we described in the previous chapter, our VULCAN application (See Chapter 4) can be used to evaluate all known vulnerabilities that affect the Cloud system of interest, including all components and shared technologies, as well as determine if there are known attacks exploiting these vulnerabilities and if there are vendor-provided patches to mitigate the security risks.

In addition to understanding what vulnerabilities plague the system, understanding threat risks associated with these vulnerabilities, the types of attacks that are enabled by the vulnerabilities and an overall risk score for the system helps prioritize mitigation actions and budgets. This is the motivation for our NEMESIS framework that provides for an automated tool that ranks vulnerabilities based on the types of threats they can lead to and the severity of the vulnerabilities to compute an overall risk value.

5.2. Background

5.2.1. Threat Modeling

Given an adversary threat model which assumes that an attacker is highly capable and well motivated, it is necessary to understand the goals of an attacker and the consequences of attacks that reflects our threat models. For our purpose, we will use Microsoft's STRIDE

threat model [62]. The different threat types included in the model are listed below.

- *Spoofing Identity (S)*. An example of identity spoofing is to obtain access using someone else's authentication information, such as username and password.
- *Tampering with data (T)*. Data tampering involves the malicious modifications of data. Examples include unauthorized changes made to data, such as database records, as the data is transferred between computers via an open network.
- *Repudiation (R)*. Repudiation threats are associated with actions that cannot be verified by others or when a user denies performing certain actions, and there is no way of countering the denials. Nonrepudiation refers to the ability of a system to counter repudiation threats. For example, a user who purchases an item might have to sign for the item upon receipt. The vendor can then use the signed receipt as evidence that the user did receive the package.
- *Information Disclosure (I)*. Information disclosure refers to threats that can expose information to individuals who are not authorized to such information.
- *Denial of Service (D)*. Denial of service (DoS) attacks deny access to services to authorized users, for example, by making a Web server temporarily unavailable or unusable.
- *Elevation of Privilege (E)*. In this type of threat, an unprivileged user gains privileged access and thereby has sufficient permissions to compromise or destroy the entire system. Elevation of privilege threats include those situations in which an attacker has effectively penetrated system defenses and become part of the trusted or privileged users.

Table 5.1 offers some simple and general ways of mitigating these threats [22]. In section 5.3, we show how we classify each vulnerability into the threat types in the STRIDE model.

Threat Type	Mitigation Technique
Spoofting Identity	Authentication Protect secrets Do not store secrets
Tampering with Data	Authorization Hashes Message Authentication Codes Digital signatures Tamper-resistant protocol
Repudiation	Digital signatures Timestamps Audit trails
Information disclosure	Authorization Privacy-enhanced protocols Encryption Protect secrets Do not store secrets
Denial of service	Authentication Authorization Filtering Throttling Quality of service
Elevation of privilege	Run with least privilege

TABLE 5.1. STRIDE Mitigation Techniques

5.2.2. Risk Assessment

Quantitative and qualitative risk assessment is a critical task that involves weighting accumulated measurements into a value that can be used by all concerned Cloud actors

(providers and users). Some real-time and offline risk assessment models such as DREAD [89] and Fenz's [34] work on using Bayesian models for assigning risk probabilities to vulnerabilities have been proposed. We extend Fenz's approach to provide a quantitative risk value for the entire Cloud computing system.

The advantage of using Bayesian threat probability determination [34] is that it gives the risk manager a methodology to determine the threat probability in a formal way. The methodology is illustrated in Figure 5.2 and each building block has its calculation schema fully documented. In addition to the Bayesian probability model, we rely on our vulnerability ontologies that define relationships and dependencies among the vulnerabilities, as well as known attacks and availability of mitigations (or patches).

Our model can be easily adapted to change how the risk probabilities are assigned based on how an organization views the different types of STRIDE threats. Moreover, the model can use both known and predicted vulnerabilities (calculated using our prediction models described in the previous chapter) to provide risk probabilities due to known and unknown vulnerabilities. Finally, in addition to providing a risk probability assessment for a given computing system, we evaluate alternate configurations for each component of the system and identify possible component configurations (or software release versions) that reduce the overall risk probabilities. The next section describes how our tool automates the process of calculating security threat risk probabilities.

5.3. Architecture

5.3.1. Design

Our NEMESIS architecture design is illustrated in Figure 5.1. Threat models and a vulnerability assessment framework are the main pillars of our architecture. We have devised two implementations for assessing cloud computing assets. To find a rich amount of security analytics auto-generated by our two implementations (Threat Probability Estimator and Threat Classification), we built four lightweight middleware applications (Risk Estimator, Severity Ranking Engine, Exploitable Vulnerabilities Generator and Suggested Configurations Generator). For a given cloud configuration, these applications produce aggregated

risk indicators, severity ranking of threat types, exploitable vulnerabilities evaluations (i.e., vulnerabilities that have known attacks), and suggested new configurations to reduce overall risk probabilities.

In this section, we describe the incorporation of the models and framework and their implementation into NEMESIS and present a high-level view of our middleware applications algorithm designs.

Our NEMESIS architecture will require the cloud assets' configuration details for performing threat modeling and risk analysis tasks. The supported formats of cloud configuration details with our architecture are either contained within an ontology knowledge base file generated by our LEGOS tool (presented in Chapter 3, Section 3.4.2) or manually built by the customer and fed through a web portal.

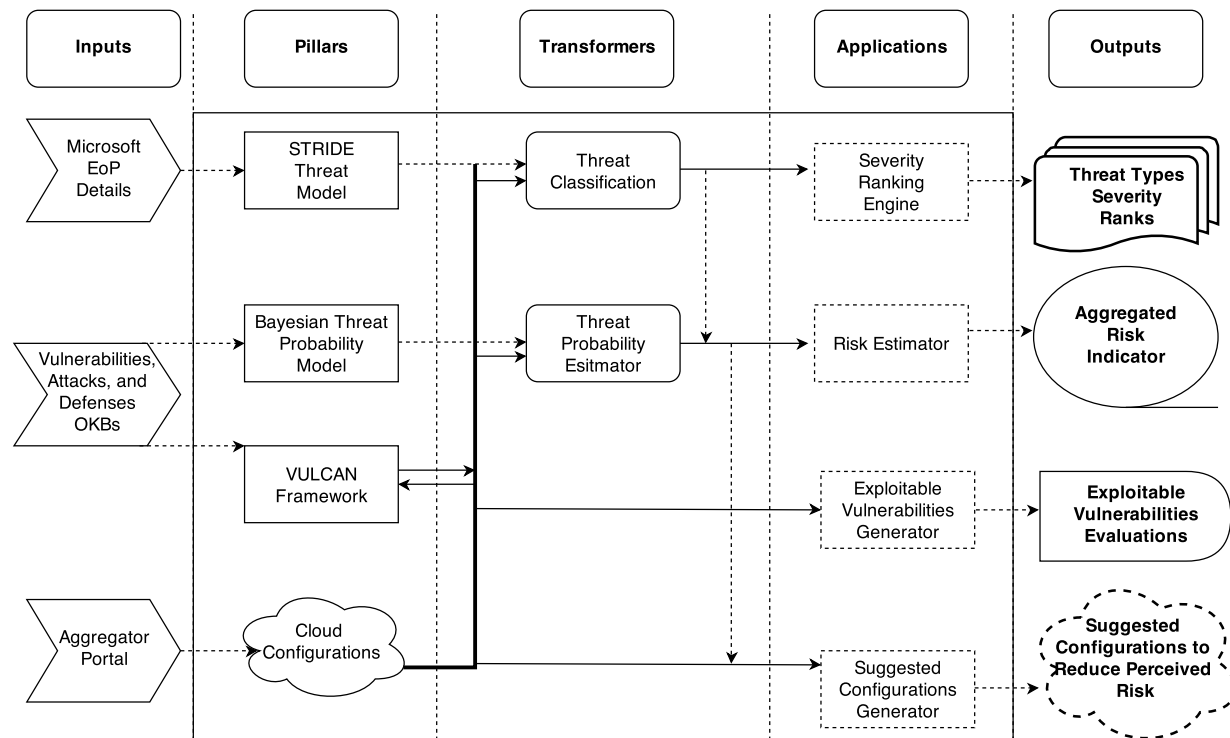


FIGURE 5.1. NEMESIS - Architecture

5.3.1.1. VULCAN Framework

The VULCAN Framework [92] (presented in Chapter 4, Section 4.2) plays a significant role in our architecture, as a source of security information used to generate various analytics. The Ontology Knowledge Bases (OKBs) generated by VULCAN (backend IKAWAFARM tool presented in Section 3.4.1) contain rich information about the vulnerabilities, attacks, and defenses associated with IT products.

NEMESIS uses the VULCAN OKBs to identify relevant information about the IT assets contained in the cloud system under evaluation. For each entity we want to see if there exists:

- Any vulnerability; if found return the vulnerability identifier, description, and severity ranking (e.g., NVD [85] CVSS severity score)
- Any attack; if found, return the exploit identifier and description
- Any defense; if found, return the mitigation identifier and recommendation statement from the vendor

5.3.1.2. STRIDE Threat Model

At present we use the STRIDE model for classifying security threats [62]. We automatically classify vulnerabilities into one of the STRIDE attacks that are possible when the vulnerability is exposed. The classification scheme automates STRIDE's Elevation of Privilege (EoP) card game [23]; where the EoP card game helps clarify the details of threat modeling and examines possible threats to software and computer systems. This approach is usually achieved manually by challenging other developers to assign STRIDE threat types relevant to the target system. However, we automated the classification using the Algorithm 4 below.

5.3.1.3. Bayesian Threat Probability Model

For the given cloud configurations, one approach to managing all threat types that could be found and classified into different threat types is to rank the vulnerabilities and threats based on their prevalence and severity risk score. It is also possible to rank the

Algorithm 4: Threat Classification**Data:** Cloud Configuration's Entities**Result:** Threat Types per each Entity - Vulnerability Pair**for** *Entity in Cloud Configuration* **do** **Invoke** *Nemesis's Vulcan Framework* instance and pass the *Entity name* **return** Relevant found Vulnerabilities details **for** *Each Found Vulnerability* **do** **Perform** *A Similarity Test to all EoP Threat Types descriptions* call, with
 Entity's Vulnerability Description as a parameter **return** Threat Types's Similarity Scores **Perform** *A Classification Task*, with *Threat Types's Similarity Scores* as a
 parameter **return** Threat type that best suit the given Cloud's Entity - Vulnerability Pair **end****end**

threats based on their significance to a specific IT installation or business model. We realize this by using a probability model to estimate the risk probability with each vulnerability and the threat type it exposes the system to, and aggregate these individual probabilities into a single value using our Bayesian-based algorithm described below.

We use Fenz's Bayesian model [34] in Figure 5.2. We adapted this model for our purpose as described in Algorithm 5 using the Bayesian Network Structure, and equations are shown in Figures 5.2 and 5.3 with a few extensions to accommodate our cloud's threat ranking needs. For example, the variables equations are grouped into Threat Variable, Intermediate Vulnerability Variable, Vulnerability Variable, Attacker Variable, Control Combination Variable, and Control Variable, and each has some dependency on the others.

5.3.1.4. Nemesis's Lightweight Applications

- Risk Estimator application high-level design is detailed via Algorithm 6
- Severity Ranking Engine application high-level design is detailed via Algorithm 7

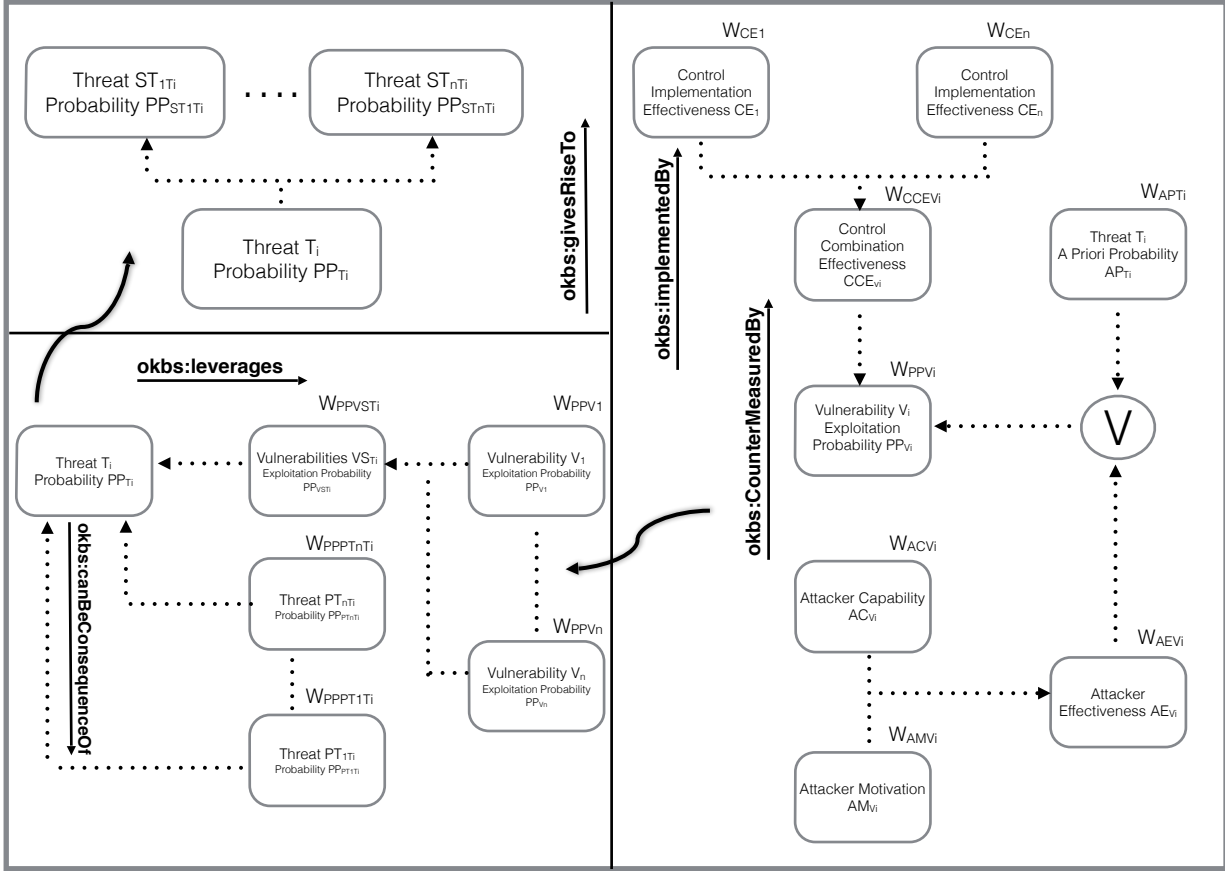


FIGURE 5.2. Utilizing OKBs for the Bayesian Threat Probability Determination

- Exploitable Vulnerabilities Generator application high-level design is detailed via Algorithm 8, and
- Suggested Configurations Generator application high-level design is detailed via Algorithm 9

5.3.2. Implementation

We have designed and implemented a prototype tool for the NEMESIS architecture. Our NEMESIS tool can be used as a standalone command line or as a web application. A preview of the NEMESIS architecture web application is shown in Fig. 6.23.

The NEMESIS tool user interface shown in Fig. 6.23 reflects its architecture design (See Fig. 5.1). The NEMESIS web application leverages LEGOS (See Fig. 6.7) and VULCAN (See Fig. 6.9) tools as its direct input to automate the collection of given cloud

$\overline{PP}_{T_i} = \overline{PP}_{VS_{T_i}} * W_{PP_{VS_{T_i}}} + \sum_{j=1}^n (\overline{PP}_{PT_{jT_i}} * W_{PP_{PT_{jT_i}}}) \quad (1)$	$i, j, n \in \mathbb{Z}^+$
$\overline{PP}_{VS_{T_i}} = \sum_{i=1}^n (\overline{PP}_{V_i} * W_{PP_{V_i}}) \quad (2)$	W^* : Associated weights \overline{PP}_{T_i} : Posterior threat probability of threat i
$\overline{PP}_{V_i} = \overline{CCE}_{V_i} * W_{CCE_{V_i}} + \overline{AE}_{V_i} * W_{AE_{V_i}} \quad (3)$	$\overline{PP}_{VS_{T_i}}$: Vulnerability combination exploitation probability of threat i $\overline{PP}_{PT_{jT_i}}$: Probabilities of T_i 's predecessor threats
$\overline{PP}_{V_i} = \overline{CCE}_{V_i} * W_{CCE_{V_i}} + \overline{AP}_{T_i} * W_{AP_{T_i}} \quad (4)$	\overline{PP}_{V_i} : Exploitation probability of vulnerability i
$\overline{AE}_{V_i} = \overline{AC}_{V_i} * W_{AC_{V_i}} + \overline{AM}_{V_i} * W_{AM_{V_i}} \quad (5)$	\overline{CCE}_{V_i} : Control combination effectiveness in the context of vulnerability i \overline{AE}_{V_i} : Attacker effectiveness in the context of vulnerability i
$\overline{CCE}_{V_i} = \sum_{i=1}^n (\overline{CE}_i * W_{CE_i} * Re_{CE_i}) \quad (6)$	\overline{AP}_{T_i} : A priori threat probability of threat i \overline{AC}_{V_i} : Attacker capabilities in the context of vulnerability i \overline{AM}_{V_i} : Attacker motivation in the context of vulnerability i \overline{CE}_i : Control implementation effectiveness Re_{CE_i} : Binary indicator

FIGURE 5.3. Equations for an Ontology Based and Bayesian Threat Probability Determination – Approach

system configurations for assessment and the generation of a vulnerability-index (using our cyber threat data ontology knowledge base presented in Chapter 3, Section 3.3.2) for threat modeling tasks. Once all “Input” details are provided to NEMESIS, one can then start exploring the results generated automatically, and tailored to the user-provided cloud system configurations via the “Threat Modeling” and “Risk Assessment” navigation portals.

The NEMESIS web application is capable of assessing multiple cloud systems, and by default, the recently provided input details will be automatically modeled, and its threat assessment details can be explored via their relevant portals such as:

- Threat Modeling
 - *Exploitable Vulnerabilities*: List of vulnerabilities that have at least one publicly known exploit that affects the assessed cloud system.
 - *Ranked Threat Types*: STRIDE based threat types (or threats using a STIX

like classification illustrated in Fig. 3.5) list ordered by the severity of each threat type based on the prevalence of vulnerabilities of the given threat type and severity of the vulnerabilities.

- *Threat Mitigation*: A detailed course of action to mitigate each identified threat type instance.
- *Validate*: A guided service to help a user track the mitigation plan of the NEMESIS identified threats.
- Risk Assessment
 - *Risk Score*: An aggregated quantitative risk score for all identified threats.
 - *Recommendations*: Alternate cloud system configurations with a reduced perceived risk value.
 - *Risk Assessment Report*: A summary report similar to one rendered by the VULCAN web application shown in Fig. 4.5) of all key insights found by NEMESIS for the given cloud system for assessment.

The NEMESIS architecture is implemented as a Python project, where we have implemented each of our algorithms as Python modules, along with necessary middleware programs and frameworks to provide an execution environment for our tool. Similar to our VULCAN web application, we have implemented the NEMESIS web application using some third-party software including:

- Django web framework [38]
- Bootstrap framework [21]
- Protege: an ontology editor and framework for building intelligent systems [81]
- AllegroGraph semantic graph database [1]
- Gruff: a grapher-based triple-store browser for AllegroGraph [60]

Algorithm 5: Threat Probability Estimator

Data: Cloud Configuration's Entities and OKBs graphs

Result: A list of Threat Probabilities Values for all given Cloud's entities

for *Entity in Cloud Configuration* **do**

Invoke *Nemesis's Vulcan Framework* instance and pass the *Entity name and relevant OKB graph*

return Relevant found Vulnerabilities identifiers

for *Each Vulnerability identifier* **do**

Invoke *controlVariable* subroutine, with *This Vulnerability identifier and relevant OKB graph* as a parameter

return A qualitative scale of this control for the given vulnerability, its description and a binary value of whether this control is active

Invoke *ControlCombVariable* subroutine, with *controlVariable outputs and other variant's similar data of This Vulnerability Identifier* as parameters

return A qualitative scale and its quantitative value

Invoke *AttackerVariable* subroutine, with *This Vulnerability identifier and relevant OKB graph* as parameters

return A qualitative scale and found Exploit Description

Invoke *vulnerabilityVariable* subroutine, with *ControlCombVariable and AttackerVariable outputs* as parameters

return A quantitative scale

 Append each Vulnerability identifier's vulnerabilityVariable output into a list

end

Invoke *intermediateVulnerabilityVariable* subroutine, with *A list of vulnerabilityVariable outputs* as parameters

return A quantitative scale

Invoke *ThreatVariable* subroutine, with *intermediateVulnerabilityVariable outputs and a list of aPriori Threat Probabilities Values* as parameters

return A quantitative scale

 Append each Vulnerability identifier's ThreatVariable output into a list

end

Algorithm 6: Risk Estimator

Data: A list of Threat Probability Values for all given Cloud's entities and their weights

Result: Aggregated Risk Indicator

for *Input Data* **do**

Invoke *an Aggregator* subroutine and pass the *Input Data* as parameters

return Aggregated Quantitative Scale

end

Algorithm 7: Severity Ranking Engine

Data: Cloud Configuration's Entities and OKBs graphs

Result: Threat Types Severity Ranks

for *Entity in Cloud Configuration* **do**

Invoke *Nemesis's Vulcan Framework* instance and pass the *Entity name and relevant OKB graph*

return Relevant found Vulnerabilities identifiers

for *Each Vulnerability identifier* **do**

Invoke *Threat Classification* module, with *This Vulnerability identifier and relevant OKB graph* as a parameter

return Perceived Entity - Vulnerability's Threat Type

Invoke *SeverityScore* subroutine, with *This Vulnerability Identifier* as parameters

return A quantitative severity score

 Append the found severity scores per each vulnerability into a list of threat type class lists

end

end

for *Lists of severity scores of threat type classes* **do**

Compute a new list of average severity scores for threat type classes and return it

end

Algorithm 8: Exploitable Vulnerabilities Generator**Data:** Cloud Configuration's Entities and OKBs graphs**Result:** Exploitable Vulnerabilities Evaluations**for** *Input Data* **do** **Invoke** *Nemesis's Vulcan Framework* instance and pass the *Entity name and Vulnerability OKB graph* **return** Counts of relevant found vulnerabilities, their identifiers and Active/Passive state binary value **for** *Each vulnerability identifiers* **do** **Invoke** *Nemesis's Vulcan Framework* instance and pass the *Entity name and Attack OKB graph* **return** Count of found Exploits

Append this Exploits Count into a list

end**end****for** *All counted list's values* **do** **Compute** their sum with respect to their targeted vulnerabilities and return the Exploitable Vulnerabilities Evaluations**end**

Algorithm 9: Suggested Configurations Generator

Data: Cloud Configuration's Entities and OKBs graphs

Result: Suggested Configurations to Reduce Perceived Risk

for *Input Data* **do**

Invoke *Nemesis's Threat Probability and Risk Estimator* module and pass the
 Input Data

return Aggregated Risk Indicator for this Cloud Configuration Profile and store
 this result for future comparison

Invoke *An Evaluator* module and pass the *Current Cloud Configuration Profile*

return All other alternative Cloud Configuration Profiles made of pre and post
 releases of the first Profile's entities

for *Each generated Cloud's Profile* **do**

 | **Compute** its Aggregated Risk Indicator and store it

end

Perform *A systematic* ranking of each *Cloud Configuration Profile and its Risk
 Indicator*

return The best optimal profile with a lower risk indicator and suggest it

end

5.4. Experiments and Evaluations

5.4.1. Experiments

To validate and evaluate our NEMESIS [91] approach for risk quantification, we have designed a cloud environment and deployed its services using an OpenStack [95] cloud operating system. For this OpenStack deployment, we used DevStack [41] and deployed one of the OpenStack version [42].

Our simple OpenStack deployment system is live with nova-compute, cinderv2-volumev2, novav3-computev3, s3-s3, glance-image, heat-cloudformation, cinder-volume, ec2-ec2, heat-orchestration, and keystone-identity services to power OpenStack's Compute and Orchestration projects. For example, we can leverage Glance and Nova services if we want to deploy any flavored instance running on the Ubuntu, Fedora, Centos, RedHat, or Windows operating system. Also, for various cloud applications, we can leverage the Heat service which in turn relies on other services.

On a Dell PowerEdge T620 server with 24 cores and 64 GB of RAM and 2TB of storage, running Ubuntu:* and using VirtualBox:* [63], we created an Ubuntu:* virtual machine for OpenStack-DevStack single node deployment powered with 10 CPUs, 40 GB of RAM and 400 GB of storage. Upon successful OpenStack deployment, the QEMU:* hypervisor is used to support the compute services.

Via the OpenStack dashboard, we can instantiate a number of instances such as *Ubuntu:**, *Fedora:**, and *Centos:** via the Compute project and some stacks via the Orchestration project using open-source based heat-templates [95] such as *Word Press Native*, *Chef Server* and *Nova Instance With Cinder Volume Native*.

A sample look at some IT products (* taken randomly from a pool of various releases of products that meet our experiment criteria) that are used to support our cloud infrastructure and services are shown in Table 5.2 which are in turn used to evaluate our NEMESIS architecture.

IT Products	Description
Ubuntu:12.04	- Host and Guest VM OS
VirtualBox:3.2	- Virtualization Product
Grizzly 2013.1.1	- Cloud OS
Ubuntu:12.10	- Cloud image OS
Fedora:17	- Cloud image OS
Centos:6	- Cloud image OS
WordPress:3.0.3	- Web Software
MySql:5.5.29	- RDBMS
RabbitMQ:3.3.5	- AMQP
Qemu:1.3.0	- Hypervisor

TABLE 5.2. Our Sample Cloud Configuration

5.4.2. Evaluations

In this section, we present our findings given the sample of cloud configurations shown in Table 5.2 that is fed to our NEMESIS tool.

5.4.2.1. Risk Estimator Application

The given cloud configuration that is shown in Table 5.2 is evaluated via this application, and the tool estimates the aggregated risk as *31.93%* of severity. This score is based on our Bayesian model described in Section 5.3, and is based on the severity of each vulnerability, the existence of known attacks and known patches: thus this score indicates an aggregated probability of potential security attacks.

5.4.2.2. Severity Ranking Engine Application

For a threat focused security approach, the given cloud configuration that is shown in Table 5.2 is ranked based on the application design principle illustrated in Algorithm 7. For instance, Spoofing is the most severe threat facing this cloud configuration, and Information Disclosure is the least severe.

Threat Types	Severity Rank (0-10)
Spoofing	4.01
Tampering	1.97
Repudiation	1.23
Information Disclosure	0.86
Denial of Service	3.28
Elevation of Privilege	1.13

TABLE 5.3. Threat Types - Severity Rank Evaluations

5.4.2.3. Exploitable Vulnerabilities Generator Application

For the given cloud configuration shown in Table 5.2, this application produces the exploitable vulnerabilities metric for each cloud configuration entity as illustrated in Table 5.4. In this case, two cloud entities stand out where Fedora:17 has a value of 1 and WordPress:3.0.3 has a value of 4. These scores are high and indicate the need for an immediate action leading to patching of these software systems. A score of 0 indicates that there are no known attacks at this time, but these software systems still contain vulnerabilities.

5.4.2.4. Suggested Configurations Generator Application

Our tool then explores several possible configurations (using alternate versions of the different software systems) and selects a configuration that results in a lowered overall risk score. For the system at hand (shown in Table 5.2), our tool recommended an alternate configuration shown in Table 5.5, and the aggregated risk with this suggested configuration is *25.88%*.

5.4.2.5. Challenges

While our NEMESIS system is based on sound models and can provide useful information on estimated or perceived security risks, we identify the following limitations with the current implementation of NEMESIS.

IT Product	Vulnerabilities Count	Exploitable Vulnerabilities Metric
Ubuntu:12.04	97	0
Virtualbox:3.2	2	0
Grizzly:2013.1.1	1	0
Ubuntu:12.10	87	0
Fedora:17	10	1
Centos:6	7	0
Wordpress:3.0.3	32	4
MySql:5.5.29	48	0
Rabbitmq:3.3.5	0	0
Qemu:1.3.0	8	0

TABLE 5.4. Exploitable Vulnerabilities Metric Evaluations

- The information used by NEMESIS (via VULCAN) is based on known vulnerabilities, attacks, and defenses that are provided by open-source communities, and often, the information is not presented in a standardized format, making automated extraction of the information difficult.
- There is a wealth of information on vulnerabilities, but the information on countermeasures is very limited and often not available in open sources (requiring one to consult vendors for such information).
- In finding alternate configurations, we rely on version numbering schemes. However, there is a lack of consistency in the version numbers, and sometimes vendors change the numbering scheme for newer releases, making it difficult to differentiate between older and newer versions of software systems.

IT Products	Pre- and Post- Releases Count
Ubuntu:13.04	22
Virtualbox:4.0.20	75
Grizzly:2013.1.4	8
Ubuntu:13.04	22
Fedora:18	15
Centos:6	1
Wordpress:3.7	121
MySql:6.0.10-bzr	366
Rabbitmq:3.3.5	0
Qemu:2.0.0	78

TABLE 5.5. Alternative Recommended Cloud Configuration

5.5. Summary

In this work, we have proposed and implemented NEMESIS, a novel automated risk assessment architecture for cloud systems. Our proposed architecture design principle revolves around collecting measurements about what type of known vulnerabilities exist for the given Cloud’s assets; how they can be exploited and how they can be mitigated; then using them toward the generation of a set of customized outputs such as: Aggregated risk estimated value, Exploitable vulnerabilities metric evaluations, Threat types - severity rank evaluations, and New recommended Cloud Configurations with a lower perceived risk along with a detailed summary of relevant NEMESIS evaluation data.

Our approach to assessing the risk of cloud systems leverages sound mathematical models powered by evolving cyber threat data, and auto-generated vulnerability-indexes ontology knowledge bases (generated by our IKAWAFARM and VULCAN tools presented in Sections 3.4.1 and 4.2). The NEMESIS architecture is designed to be scalable to support

real-life IT systems. Its components can be interchanged or extended to support new use cases. For instance, NEMESIS can be extended to include vulnerability information of the assessed IT systems using our predictive models which assess the complexity and maturity of a software product and predict the number of unknown vulnerabilities that are yet to be found and disclosed (See Section 4.4)) to update the IT system's estimated security risk dynamically. Also, NEMESIS can be used to estimate the exploitation risk of a given cloud system's ranked vulnerability findings based on a probabilistic attack graph model (See Section 4.3).

In the next Chapter 6, we present our novel system called COCKATOO. COCKATOO is a holistic suite of IT system security tools for threat assessment and mitigation. The COCKATOO system places the NEMESIS architecture at its center stage, including all of NEMESIS dependency and extension tools (LEGOS, IKAWAFARM, VULCAN, SWEEP, and PREDICTION) presented in the previous Chapters 3, and 4.

CHAPTER 6

COCKATOO SYSTEM

6.1. Overview

Organizations and individuals are falling prey to targeted cyber-attacks that result in damaging consequences such as data breaches, loss of revenues, services disruption, and accounts take-over.

More users can prevent or mitigate most cyber attacks. However, we know that Information Technology (IT) products (i.e., hardware, software, and applications) are vulnerability prone and can get compromised by malicious actors.

Currently, there are over 100K publicly known vulnerabilities that affect various computing/information systems. These vulnerabilities are reported and published within the National Vulnerability Database (NVD) [85]. Hundreds of new vulnerabilities are being reported weekly and added to the NVD. This alarming rate of new vulnerabilities that affect IT products poses a real challenge to an organization and individuals who may not even be aware of these vulnerabilities that could lead to serious attacks.

In this work, we have created COCKATOO, a novel holistic system that empowers organizations and individuals with the tools to assess and mitigate vulnerability-based threats. COCKATOO is an ontology-based suite of security tools for assessing and mitigating threats to IT systems. COCKATOO relies on a knowledge-base which can represent security vulnerability information about an instance of an IT system. This permits reasoning about threats faced by the IT system and offers actionable mitigation approaches.

COCKATOO system leverages a number of our previous works presented in Chapters 3, 4, & 5. In this Chapter, I present how these previous works are integrated into a single framework called COCKATOO.

- **IKAWAFARM** is a tool for designing scalable ontologies and their semantic knowledge graphs (including their semantic query APIs) for cybersecurity threat data and cloud computing domains. Within IKAWAFARM, we have defined vulnerability on-

tologies (and populated them), which are then used by our VULCAN tool which performs vulnerability assessment tasks for IT systems. In addition, IKAWAFARM enables us to define other ontologies for a given specific application.

- **LEGOS** is a tool that collects telemetry data of a computing host (i.e., physical/virtual/container machine) and stores them in a designated semantic graph database.
- **VULCAN** [92] provides an on-demand vulnerability assessment web application.
- **HUMMING** is a semantic natural language processing Chatbot that works with VULCAN to provide semantically rich responses to user queries.
- **NEMESIS** [91] provides an on-demand threat assessment and mitigation web application.
- **SWEEP** automates the process of software complexity metrics generation and analysis for any given IT software product and its vulnerability history. Its output is used for building a machine learning model to predict hidden or yet unknown vulnerabilities.
- **PREDICTION** tool uses the data gathered by SWEEP to predict hidden vulnerability and to implement our previously published models [90].

The rest of this chapter is presented as follows: In Sections 6.2-6.3, we present our COCKATOO toolkit’s architecture and implementation. In Sections 6.4, we present our COCKATOO system’s evaluation. We then present our COCKATOO system discussion and concluding remarks in Section 6.5.

6.2. Architecture

COCKATOO is a knowledge base system driven by ontologies for describing security vulnerabilities of IT systems.

Vulnerability-based attacks on an IT system can lead to damaging consequence for individual users and organization. Assessing these types of threats can be challenging because most IT products (e.g., softwares, services, etc.) are complex (leveraging a stack of other softwares/services dependencies), where any vulnerability in the one product can open doors

to malicious actors, which can cripple not only that product but all products that depend on it.

The goal of COCKATOO system is to empower novices, experienced users, and teams by using a set of tools to assess the security posture of their assets including ways to mitigate vulnerability-based threats. Components of COCKATOO tools have been introduced in Section 6.1. Figure 6.1 illustrates the COCKATOO tools' dependency upon their relationship with other tools. In this section, we present the architectural design of each of our COCKATOO system's tools.

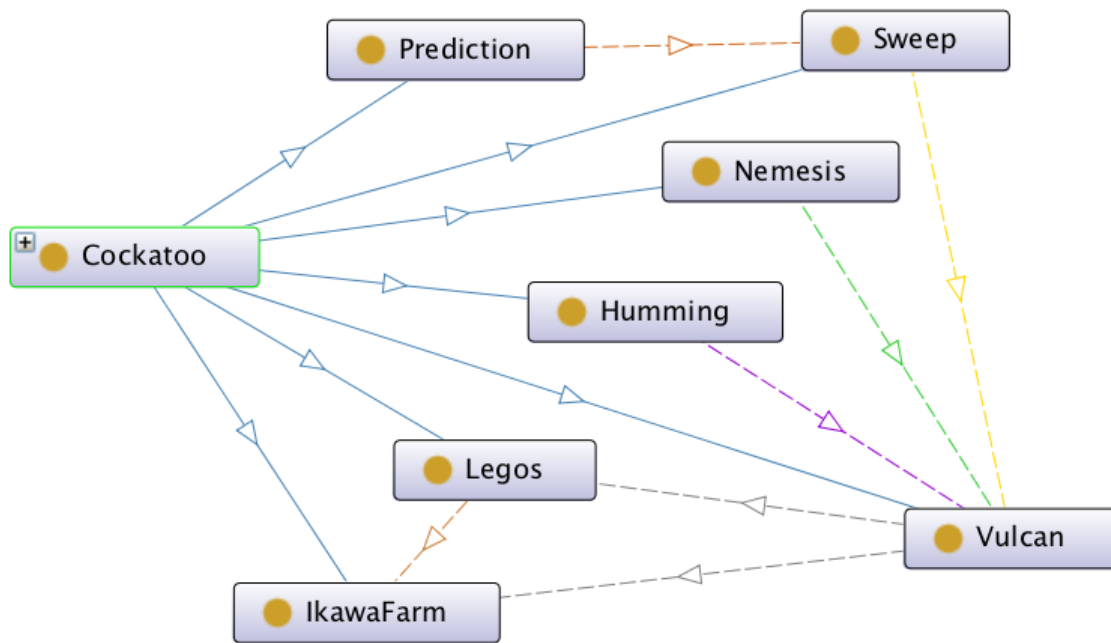


FIGURE 6.1. COCKATOO Tools Dependency Graph

The **IKAWAFARM** tool is at the core of our COCKATOO system. Its architectural design is set around these goals:

- Modeling and representing cybersecurity and cloud computing domains using an ontological approach.
- Designing and exposing a semantic API to query and reason about the generated ontology knowledge bases (OKBs).
- Providing an execution environment to maintain **IKAWAFARM**'s ontology knowl-

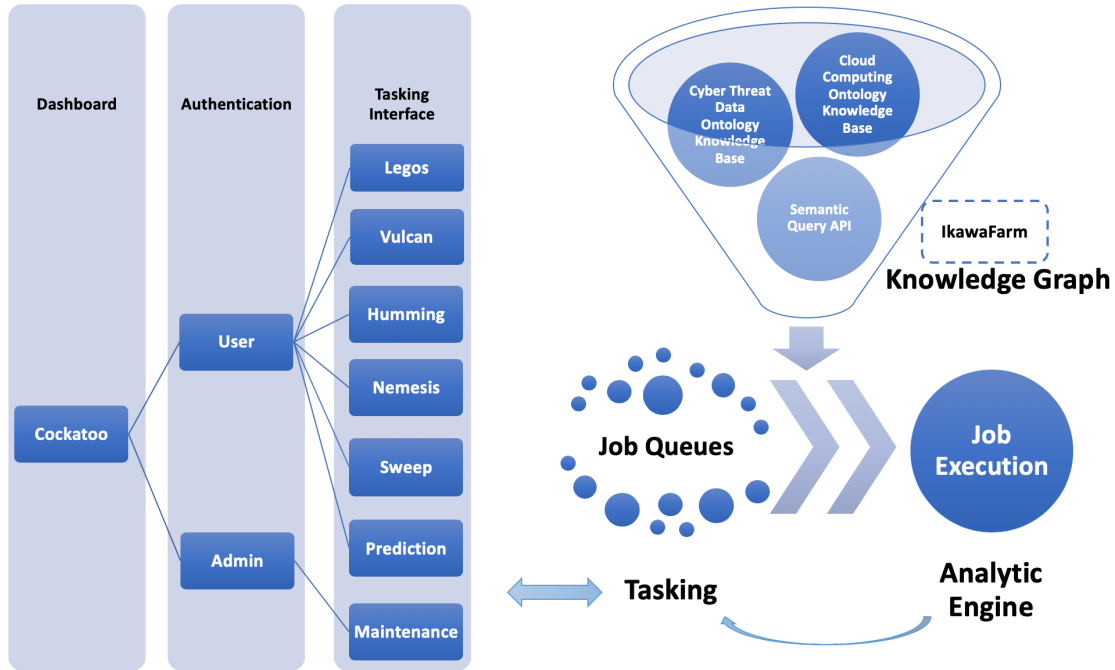


FIGURE 6.2. COCKATOO System Workflow

edge bases, and to power other COCKATOO's tools and services. Figure 6.1 shows how all COCKATOO's tools depend on the IKAWAFARM tool directly or indirectly through other tools.

In the scheme of the COCKATOO system, the **IKAWAFARM** tool sits in the background as shown in Figure 6.2 and supports extensible, evolving ontologies and knowledge bases reflecting the latest cyber threat data bases for cloud computing systems.

The **LEGOS** tool is a critical component of the COCKATOO system as it captures the details of an IT system's components. Its goals are:

- To leverage a pre-defined ontology model for the cloud computing domain (generated by the **IKAWAFARM** tool) to automate the knowledge base generation for any given deployed IT system (e.g., cloud system deployment).
- Automation is at the core of this tool. The LEGOS interface provides end-users various features to assist them in representing their IT system's assets (which could be vulnerable to exploitation by a motivated malicious actor).

VULCAN is at the center of the COCKATOO system. Its architectural design is

set around these goals:

- To leverage the **LEGOS** tool to create an ontology knowledge base (OKB) of a given IT system for vulnerability assessment tasks.
- To provide an interface that enables end-users to task **VULCAN** to index the modeled IT system.
- To allow the indexing task to extend the given IT system's OKB with information about which assets are vulnerable to exploitation, including all discovered vulnerability details (e.g., vulnerability information, vulnerability-exploit details, and vulnerability-patch details).
- To leverage the **IKAWAFARM** tool as the source of ground truth about the modeled and represented cyber threat data-feeds.
- To provide an interface to end-users to render a vulnerability assessment report using a generated IT system vulnerability index.

The **HUMMING** tool augments the capability of the **VULCAN** tool towards assisting end-users to gain insights into the security posture of their assessed IT system(s). Its architectural design is set around these goals:

- To provide a dialogue based interface to interact with end-users.
- Using natural language processing, to interpret user-provided questions in natural language (e.g., English) and forwards its output to a back-end semantic querying and reasoning service.
- To leverage ontology knowledge bases generated by our **VULCAN** and **LEGOS** tools as ground truths when providing answers to user's questions.

The **NEMESIS** tool automates the tasks of modeling threats and risk assessment of a given organization IT system. Its architectural design is set around these goals:

- Leverage **VULCAN** tool and its dependency tools (see Figure 6.1) towards acquiring a vulnerability index knowledge base for a given organization's IT system.
- Using a cached or rendered on-fly vulnerability index of the given IT system, provide interfaces for various threat modeling and risk assessment tasks.

- Render a summary report of key findings of the assessed assets, in terms of the estimated threat probability/risk that each threat exposes the assessed IT system to a motivated malicious actor, along with recommendations to mitigate the perceived risk.

The **SWEEP** tool automates the process of building a rich dataset that captures various software metrics that give insight into how a given software product has matured and where it currently stands. Its architectural design is set around these goals:

- To provide an interface to obtain the source codes of all releases of a given software product, and automate the software complexity metrics generation using a back-end tool called *Understand* [64].
- For the generated software complexity metrics, provide an interface to automate their analysis and extend them by querying the back-end knowledge bases of our **IKAWAFARM** tool to obtain vulnerability history of the product releases.
- To provide an interface that facilitates the datasets generation for each studied software product. Each generated dataset includes a number of features about a given software product's software complexity metrics and maturity in terms of its vulnerability history timeline (this data is fed to our Prediction tool).

PREDICTION is a machine learning tool for predicting the unknown number of vulnerabilities in a given software product. Its architectural design is set around these goals:

- To leverage the **SWEEP** tool to generate machine learning datasets for the training and inferencing tasks.
- To provide an interface to train a suitable regression machine learning model to predict the unknown number of vulnerabilities in a given software product.
- To provide an interface for inferencing, where a user can provide a new or current software release metrics (auto-generated by our **SWEEP** tool) and select the previously trained model and infer (prediction task) the number of yet to be found vulnerabilities.

- To provide an interface that enables users to load pre-trained machine learning models for various inferencing tasks.

In the next Section 6.3, we describe implementation details of these tools.

6.3. Implementation

The COCKATOO system is implemented as a web system which features a gallery of tools offering on-demand services. Each of COCKATOO's tool (IKAWAFARM, LEGOS, VULCAN, HUMMING, NEMESIS, SWEEP, and PREDICTION) is implemented as a Python package (including some third-party packages when applicable as illustrated in Figure 6.4) and exposed to end-users via a web portal shown in Figure 6.3. Also, each of COCKATOO tools can be launched from the main web portal as a standalone web application (See Figure 6.5) or as a customized service offering or as one of the features of the analytic dashboard interface.

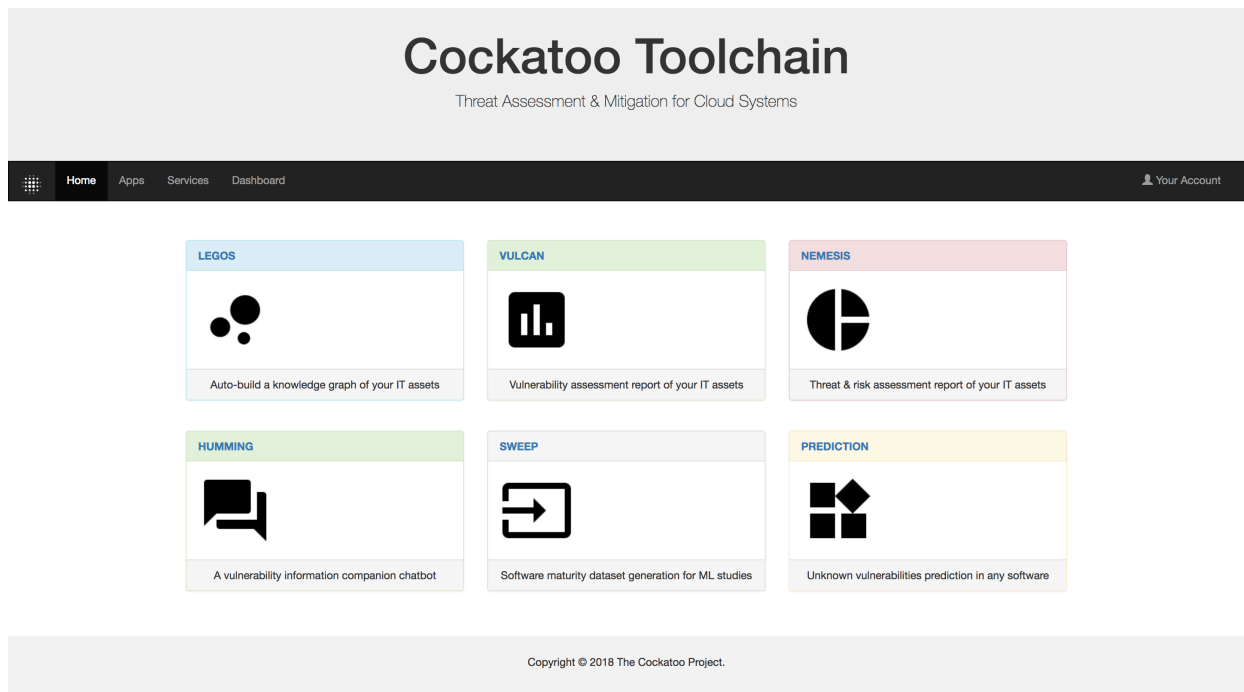


FIGURE 6.3. COCKATOO System – Web Portal

In Chapters 3, 4, and 5, each of COCKATOO tools is presented individually. In this Chapter, we focus on presenting how these tools interact with each other within COCKATOO

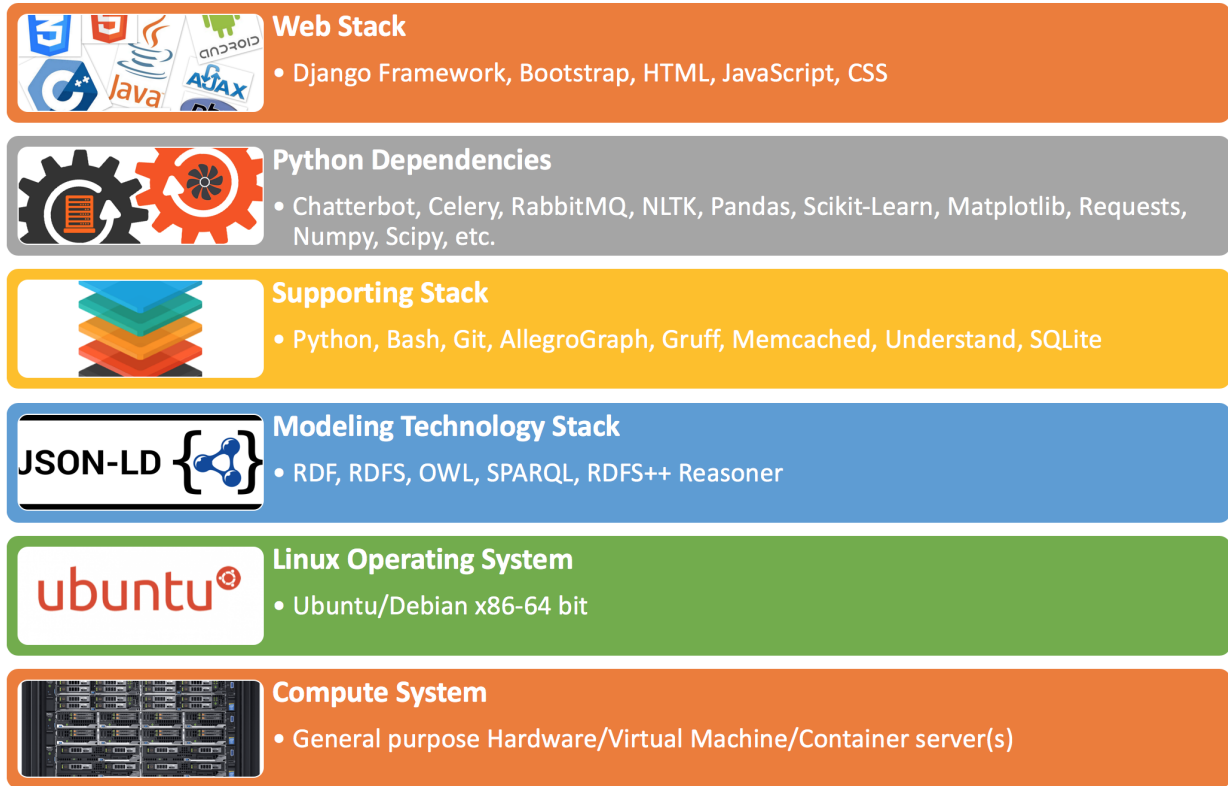


FIGURE 6.4. COCKATOO System – Technology Stack Sample

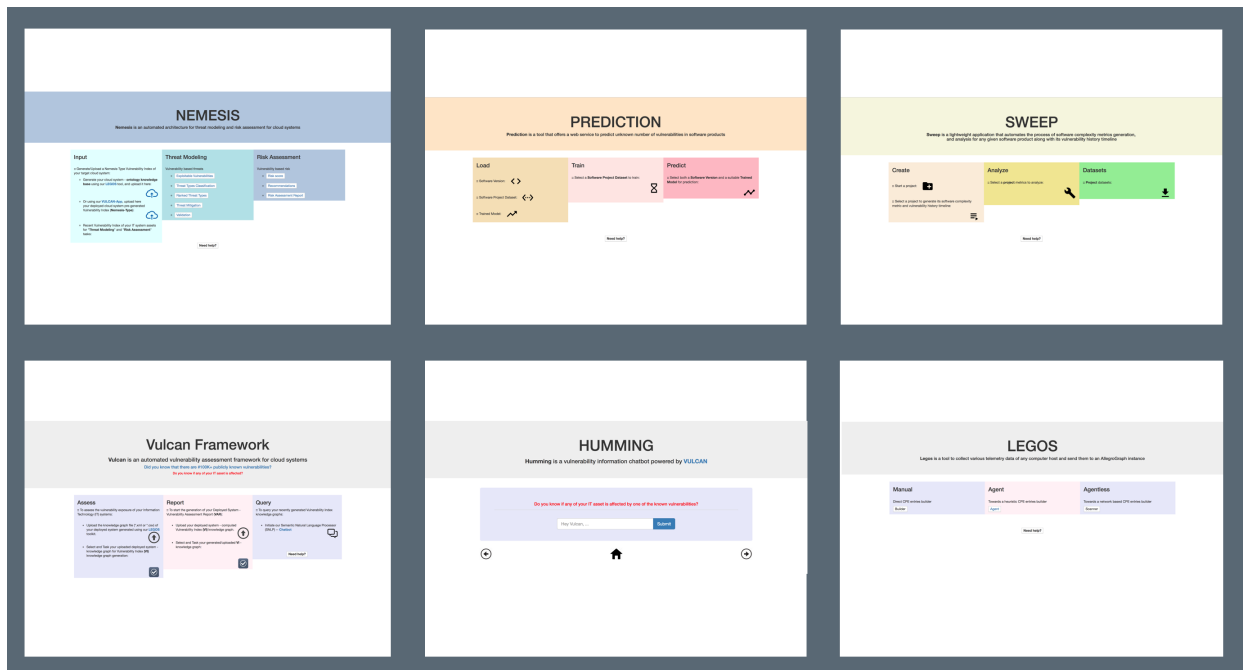


FIGURE 6.5. COCKATOO Tools – Gallery Preview

system as a holistic solution to the problem of vulnerability-based threats that affect any IT system.

The COCKATOO system comprises two views, the front-end (web portal and its tools web applications) and back-end (execution environment and background processing for all its tools). Each of COCKATOO tools is implemented to meet their architectural goals described in the previous Section 6.2.

The *IKAWAFARM* tool as part of COCKATOO system runs as a background service. It is implemented in Python to leverage a Python API of AllegroGraph [1] (our chosen semantic graph database, which supports RDF, RDFS, OWL ontology languages, and provides a reasoner) to generate and store the ontology artifacts of our modeled domains (cyber threat data and cloud computing). In addition, IKAWAFARM automates the collection of relevant data-feeds (e.g., NVD [85], and Exploit-DB [104]) and extracts the information and populates our defined ontology artifacts using our Algorithm 1 while leveraging a number of AllegroGraph instances (running in the background and self-managed) as its back-end datastores. Then, IKAWAFARM implements a number of methods to build a rich semantic API to enable other COCKATOO tools (e.g., VULCAN) to query relevant data from the generated cyber threat ontology knowledge bases. Then, the COCKATOO system implements an execution environment of IKAWAFARM and administration tools to maintain IKAWAFARM evolving ontology knowledge bases, functionalities, and integration with other COCKATOO tools.

The *LEGOS* tool as part of COCKATOO system runs as a standalone application offered through COCKATOO web portal. Also, LEGOS has a command line utility that enables it to run as a managed software agent. LEGOS is also implemented in Python. It is implemented to be able to query an operating system (for a given computing host) to collect telemetry data (e.g., deployed services, installed applications and their dependencies) and store them in a back-end datastore (AllegroGraph [1] instances) for post-processing tasks. It leverages a generic ontology model from IKAWAFARM ontology artifacts to model and represent a deployed IT system (e.g., a deployed private cloud system). More specifically, the

LEGOS tool is in charge of providing standardized data formats to model and represent a given IT system for later security threat assessment tasks by other COCKATOO tools (e.g., VULCAN), and analytics engine provided within the COCKATOO dashboard. Figure 6.7 shows LEGOS web application. LEGOS main web interface is divided into three features “*Manual, Agent, and Agentless*”. *LEGOS – Manual* interface enables the user to supply their IT system’s products information using a Common Platform Enumeration (CPE) [83] format or to input partial information that they know about their IT system, then the LEGOS tool will construct a profile of those inputs and generates an ontology representation of the given IT assets. *LEGOS – Agent* interface provides a software agent that runs on a given computing host, and collect the telemetry data automatically using an ontology model, and store them in a designated triple-store. *LEGOS – Agentless* interface prompts the user for network-based information of the user’ deployed IT system, and remotely query the system, and generate an ontology knowledge base about discovered IT assets and their configuration.

The *VULCAN* tool as a part of COCKATOO systems runs as a web application offered via the COCKATOO main web portal. It relies on the LEGOS and IKAWAFARM tools. It prompts the user to supply an ontology knowledge base of their IT system (generated by the LEGOS tool) as an input. Then, VULCAN offers a web interface to users to task their IT system for vulnerability indexing task (which is run as a background service). The vulnerability indexing task queries the backend IKAWAFARM for known vulnerabilities, exploits, and mitigation that are relevant to the user-supplied IT system data artifact; then generates a vulnerability index ontology knowledge base (OKB) (different levels of indexing are provided to the user during the process). This later vulnerability index OKB is then made available to the main VULCAN application for vulnerability assessment tasks. The user can then elect to view a vulnerability assessment report of their IT system by selecting the vulnerability index generated for the tasked system. In addition, a user can elect to use a chatbot service of VULCAN to interactively ask questions about their IT system security posture in natural language (English).

The *HUMMING* tool as part of COCKATOO systems, runs as a web-based chat-

bot service. It leverages the ontology knowledge bases generated by VULCAN to assess questions about the security posture of the IT system assessed by our COCKATOO tools (e.g., VULCAN, and NEMESIS). It also uses other ontology knowledge bases generated by IKAWAFARM and LEGOS for answering general questions. The novel approach to the chatbot implementation arises from using the ontology model concepts, properties, and relationships for the modeled domains through semantic querying and reasoning about the generated knowledge bases as a basis to answer users' questions.

The *NEMESIS* tool as part of COCKATOO systems runs as a web application offered via the COCKATOO main web portal. It leverages the VULCAN tool. It prompts the user to supply a vulnerability index ontology knowledge base of their IT system generated by VULCAN as input. Then, NEMESIS uses this vulnerability-index for automated threat modeling and risk assessment tasks. For each task, NEMESIS provides a user-friendly web interface to the users so that they can render those assessments. For the threat modeling tasks, a user can access threat information about their IT assets such as *Exploitable Vulnerabilities*, *Threat Types Classification*, *Ranked Threat Types*, *Threat Mitigation*, and *Validation*. For the risk assessments tasks, a user can access on-demand risk information about their IT assets such as *Risk score*, *Recommendations to mitigate the estimated risk*, and *Risk Assessment Report*. In the background, NEMESIS queries the backend IKAWAFARM ontology knowledge bases to determine the best alternate IT system configurations to recommend to the user to lower risks.

The *SWEEP* tool as part of COCKATOO systems runs as a web application offered via the COCKATOO main web portal. It automates the process of generating and analysis of software complexity metrics and maturity for any software product. Through SWEEP web interface, a user can create a project to be used as a repository of source code files for all known releases of a software product under study (e.g., OpenSSL [40]). Then, the user can task SWEEP to generate software complexity metrics for the created project. In the background, SWEEP uses *Understand* [64] to generate complexity metrics for the given software project. Then, SWEEP automatically forwards and makes available the generated

complexity metrics for the selected project via the web portal. The user can select the generated complexity metrics for analysis. The SWEEP tool will query IKAWAFARM backend ontology knowledge bases for vulnerability information for each software release and generates a vulnerability history timeline for the software project (its maturity). Then, SWEEP analyzes both the software complexity metrics and maturity of the product and generates a rich dataset in comma-separated values (CSV) format to be used by the PREDICTION tool or for other post-processing tasks such as data analytics.

The *PREDICTION* tool as part of COCKATOO systems runs as a web application offered via the COCKATOO main web portal. It is implemented as a web interface to assist users in training a predictive model for later inferencing about the number of unknown vulnerabilities in a given software product. The PREDICTION tool relies on the SWEEP tool to assist the users in generating datasets of a given software product (all software releases or some releases of interest) and then input them into the PREDICTION web portal for machine learning training or inferencing tasks. For the machine learning training task, a user selects their dataset of interest and select the target feature for prediction tasks (e.g., by default the tool identify the total number of vulnerability as the predictor feature), and then, a background process is initiated for an automated machine learning experiment to identify the best features from the given dataset and train a suitable learning model (for example, a *Gradient Boosting Regressor Ensemble* [66] model). A trained model will be evaluated and then saved for persistence and later usage. The user can select the trained model, along with uploaded data for software releases (generated using SWEEP for a small set of software releases of interest for prediction tasks) and initiate the inferencing task. The PREDICTION tool automatically infers the number of unknown vulnerabilities that are yet to be discovered in those targeted software releases and generates a report of those findings. In addition, our PREDICTION tool enables the user to upload other machine learning models for inferencing tasks. In some cases, software products which share a number of similarities (e.g., choice of programming language, and service offering) can interchangeably use the same machine learning models to predict the number of unknown vulnerabilities in

Language	Files	Blank	Comment	Code
IPython Notebook	20	0	0	27544
Python	163	5869	4093	13457

TABLE 6.1. COCKATOO System – Python Codebase Distribution

a chosen software release with good confidence in the prediction.

All these discussed tools are embodied in our COCKATOO system. Our COCKATOO system has been built on a general-purpose server (running a Linux operating system distribution) with sufficient processing, networking and memory resources to power the system. Table 6.1 shows a sample of our COCKATOO system codebase (of 3295 unique files); it was generated using CLOC [3] tool and highlighted the amount of generated Python codes (note that COCKATOO leveraged other programming languages for specific tasks, but their contributed codebase is minimal). For the Python codebase (and other middleware technologies), there are several Python dependencies (See. Figure 6.4) packages and third-party software leveraged for different functionality (e.g., data analytics, machine learning, caching, background task/queue(s) processing, etc.).

Two direct verticals of COCKATOO system implementation extensions will revolve around providing tailored services, and unified dashboard to track changes in the user’ IT system (throughout its life-cycle) and how the user has been applying our threat mitigation recommendations over time.

6.4. Evaluation

In this Section, we present our evaluation of COCKATOO system with a focus on three use-cases illustrated in Figure 6.6 using some example of IT systems (shown in Table 6.2).

6.4.1. IT Assets Inventory

An organization may be using any of the IT systems listed in Table 6.2 to achieve core mission goals. At any given point in time, an organization must have a system in

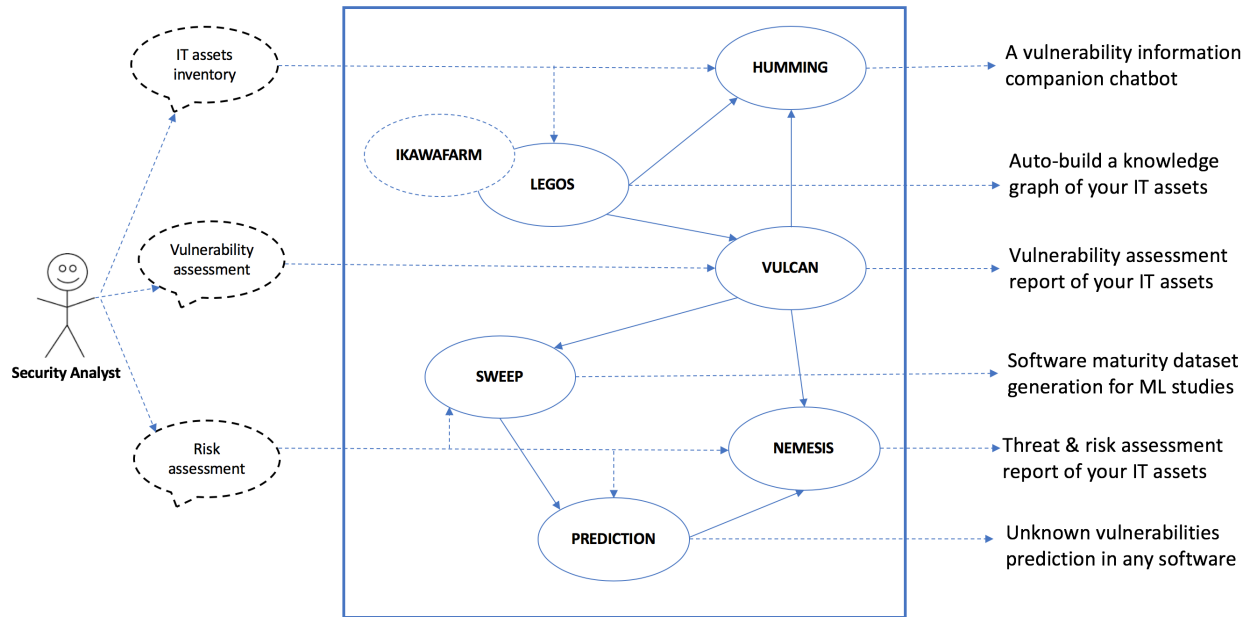


FIGURE 6.6. COCKATOO Tools – Use Cases

IT System	Description
OpenStack	A cloud OS that controls large pools of compute, storage, and networking resources throughout a datacenter
Django	A high-level Python Web framework that encourages rapid development and clean, pragmatic design
Joomla	A content management system (CMS) for publishing web content
Drupal	A content management system (CMS) for publishing web content
LAMP	A "Linux, Apache, MySQL, and PHP" (aka. LAMP) generic software stack
WordPress	A content management system (CMS) for publishing web content
Office	A Microsoft Office productivity suite
OpenSSL	A toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols

TABLE 6.2. Example of IT Systems

place that enables them to gain a clear understanding of the IT assets in their organization, their configurations, and uses. Such systems should enable the organization to analyze and implement actions to assure that their IT assets are protected against security threats and are operating in a safe and secure state. This will ensure that they are not exposing their critical information or being exploited by malicious actors to their peril.

Within our COCKATOO system, we have designed the LEGOS tool (See Figure 6.7) to assist individuals and organizations in gaining visibility into their IT assets that could be

exposed to attacks by a malicious and motivated actor.

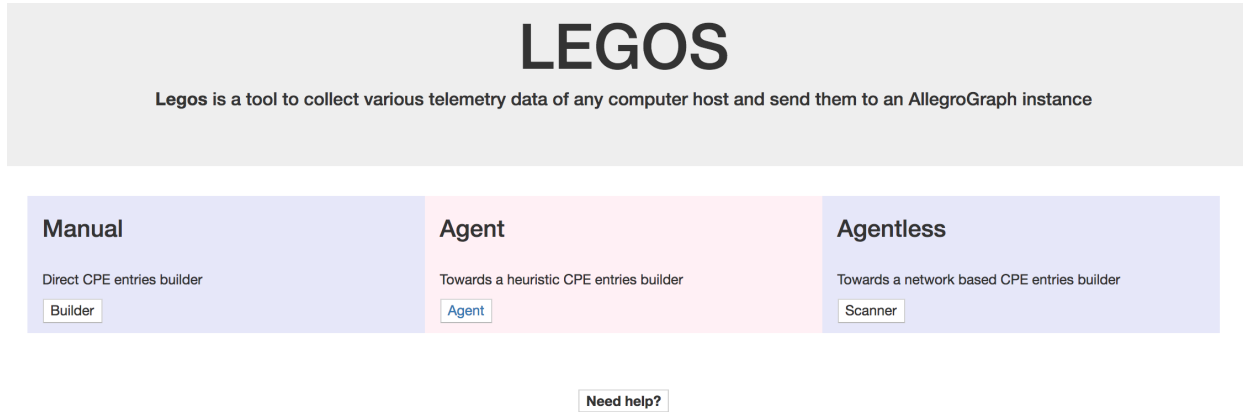


FIGURE 6.7. COCKATOO – LEGOS Tool

For individual products like *OpenSSL* [40] and *Office* [74], using our LEGOS, we can capture their representation using a standard model of representing IT products using Common Platform Enumeration (CPE) [83] standard. In this case, if a given user knows the product name and its version (e.g., “*OpenSSL 1.1.0a*”), then our LEGOS will construct its CPE representation using IKAWAFARM ontology knowledge bases (which house many instances of IT products represented in CPE format) and return its representation as “*cpe:/a:openssl:openssl:1.1.0a*”. Similarly for the “*Office 2016*” product LEGOS will return “*cpe:/a:microsoft:office:2016*” as its representation. Then, LEGOS will export their representation into a CSV file for vulnerability assessment tasks. A similar approach of representing individual products using our LEGOS manual interface (See Figure 6.7) can be done in case of a software stack like WordPress [124], see Table 6.3 for illustration. If a user is already familiar with the CPE format, he/she can simply provide this information in CSV format.

LEGOS tool has an agent feature (See Figure 6.7), which does alleviate manual processing in representing a user’ IT system. Using the WordPress example, a user may actually be running a WordPress powered site on a virtual machine instance locally or in a public cloud. In this case, there are a number of other IT products that present interdependencies for the WordPress software stack. To take into account their participation

IT Product	CPE Format
Apache 2.0.59	cpe:/a:apache:http_server:2.0.59
PHP 5.2.3	cpe:/a:php:php:5.2.3
MySQL 5.0.37	cpe:/a:oracle:mysql:5.0.37
WordPress 2.2.2	cpe:/a:wordpress:wordpress:2.2.2

TABLE 6.3. LEGOS – Bitnami WordPress V0.9 Software Stack Example

The screenshot shows the AllegroGraph WebView 6.4.3 interface. At the top, there is a navigation bar with 'Utilities | Admin | User admiral' and a 'Documentation' link. Below this is a 'Catalogs' section with a 'system' dropdown. The 'Repositories' section displays several repositories, each with a small icon and a name: Django-LEGOS, Drupal-LEGOS, Joomla-LEGOS, Lamp-LEGOS, OpenStack-LEGOS, and WordPress-LEGOS. Below the repositories, there is a 'Create new repository' section with a 'Name:' input field and a 'Create' button. Underneath that is a 'Start session' section with a 'Session specification:' input field, a help icon, and checkboxes for 'autocommit' (checked) and 'load initfile' (unchecked), followed by a 'Start' button.

FIGURE 6.8. LEGOS Agent – Datastore View of Auto-generated IT Systems OKBs

in the overall assessment, our LEGOS agent query the operating system (e.g., Linux based OS) for various telemetry data and represents them using our ontology model (See Figure 3.2) and automatically generate an ontology knowledge base of the entire IT system (in this case, the WordPress deployment in a Virtual Machine). This WordPress-OKB will be used to answer questions regarding the entire software stack and its vulnerability posture.

For really complex IT system deployment, where manual tracking of all assets is infeasible (e.g., running an OpenStack OS [95]) or CMS system (e.g., Django [38], Drupal [29] and so on), our LEGOS agent can assist in automatically bootstrapping an ontology representation knowledge base that takes into consideration all participating hosts, and what

IT System	Triples
Ansible – OpenStack Queens	24,049
Bitnami – Django 1.11.14	2,137
Bitnami – Joomla 3.8.10	2,133
Bitnami – Drupal 8.5.5	2,137
Bitnami – Lamp 7.2.8	2,228
Bitnami – WordPress 4.9.7	2,141

TABLE 6.4. Number of Triples of Deployed IT Systems OKBs – Generated by our LEGOS Agent

kind of applications and services running on them. Table 6.4 shows the number of semantic facts (triples), our LEGOS was able to collect in representing these IT systems deployments (Figure 6.8 shows a screenshot of AllegroGraph WebView [58] of some LEGOS generated and stored IT systems OKBs).

Each of these IT systems deployment ontology knowledge bases (OKBs) generated by LEGOS can be exported in any standard ontology file format such as RDF/XML and can be used directly for visualization using tools like Protege [37] and Gruff [60] (See Figures 3.9, 3.6, and 3.7 for an example of a modeled and represented OpenStack [95] private cloud system deployment). Also, our VULCAN tool can use these LEGOS generated OKBs as inputs for vulnerability assessment tasks, and our HUMMING chatbot can enable users to ask any question they may have about their IT systems’ security posture.

6.4.2. Vulnerability Assessment

Within our COCKATOO system, we can use the VULCAN tool (See Figure 6.9) for vulnerability assessment of individual and deployed IT systems. Using examples shown in Table 6.2, Figure 6.11 shows how a user can leverage LEGOS to generate ontology knowledge bases (OKB) of each IT systems, and input them into VULCAN tool (via the “Assess” navigation), then initiate Vulnerability indexing process. When it is finished, the user can then

Vulcan Framework

Vulcan is an automated vulnerability assessment framework for cloud systems

Did you know that there are #100K+ publicly known vulnerabilities?

Do you know if any of your IT asset is affected?

Assess

:: To assess the vulnerability exposure of your Information Technology (IT) systems:

- Upload the knowledge graph file (*.xml or *.csv) of your deployed system generated using our **LEGOS** toolkit.
- Select and Task your uploaded deployed system - knowledge graph for Vulnerability Index (**VI**) knowledge graph generation:

Report

:: To start the generation of your Deployed System - Vulnerability Assessment Report (**VAR**):

- Upload your deployed system - computed Vulnerability Index (**VI**) knowledge graph.
- Select and Task your generated/uploaded **VI** - knowledge graph:

Query

:: To query your recently generated Vulnerability Index knowledge graphs:

- Initiate our Semantic Natural Language Processor (SNLP) -- **Chatbot**

Need help?

FIGURE 6.9. COCKATOO – VULCAN Tool

HUMMING

Humming is a vulnerability information chatbot powered by VULCAN

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

←
🏠
→

FIGURE 6.10. COCKATOO – HUMMING Tool

select the corresponding vulnerability index OKB of the IT system and render a vulnerability assessment report (via the “Report” navigation).

Figure 6.12 shows an example of a such vulnerability report for our studied OpenStack cloud deployment example. A user can start exploring the report by selecting a section of

Vulcan Framework

Vulcan is an automated vulnerability assessment framework for cloud systems

Did you know that there are #100K+ publicly known vulnerabilities?

Do you know if any of your IT asset is affected?

The screenshot displays the Vulcan Framework Portal interface, divided into three main sections: Assess, Report, and Query.

- Assess:** This section is titled "Assess" and provides instructions: "To assess the vulnerability exposure of your Information Technology (IT) systems:". It includes a list of actions: "Upload the knowledge graph file (*.xml or *.csv) of your deployed system generated using our LEGOS toolkit." and "Select and Task your uploaded deployed system - knowledge graph for Vulnerability Index (VI) knowledge graph generation:". Below these instructions is a list of 10 example files, each with a circular icon and a timestamp, such as "okbs/passat2014_configs.csv (Sept. 15, 2018, 1:17 a.m.)".
- Report:** This section is titled "Report" and provides instructions: "To start the generation of your Deployed System - Vulnerability Assessment Report (VAR):". It includes a list of actions: "Upload your deployed system - computed Vulnerability Index (VI) knowledge graph." and "Select and Task your generated/uploaded VI - knowledge graph:". Below these instructions is a list of 10 example files, each with a circular icon and a timestamp, such as "documents/passat2014_configs-Index_Vulcan.xml (Sept. 15, 2018, 6:37 a.m.)".
- Query:** This section is titled "Query" and provides instructions: "To query your recently generated Vulnerability Index knowledge graphs:". It includes a list of actions: "Initiate our Semantic Natural Language Processor (SNLP) -- Chatbot".

Each section has a "Need help?" button and a checkmark icon at the bottom right.

FIGURE 6.11. VULCAN – Portal

interest:

- *Executive Summary* – See Figure 6.13 for an example.
- *Asset View* – See Figure 6.14 for an example.
- *Vulnerability View* – See Figure 6.15 for an example.
- *Exploit view* – See Figure 6.16 for an example.
- *Mitigation View* – See Figure 6.17 for an example.
- *Visualization View* – See Figure 6.18 for an example.
- *Next Steps* – Discusses how HUMMING and NEMESIS tools leverage VULCAN's vulnerability index ontology knowledge bases for their specialized functionalities.

For all of our VULCAN assessed IT systems, Table 6.5 presents the number of seman-

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

VULCAN - Vulnerability Assessment Report (VAR) for openstack-queens-LEGOS-Index_Vulcan.xml - Input

Table of Contents

1. Executive Summary
2. Asset View
3. Vulnerability View
4. Exploit view
5. Mitigation View
6. Visualization View
7. Next Steps



FIGURE 6.12. VULCAN – OpenStack Vulnerability Assessment Report

Vulcan Framework

Vulcan is an automated vulnerability analysis framework for cloud systems

Do you know if any of your IT asset is affected by one of the known vulnerabilities?

Executive Summary

From a total of **118168** facts, that we know about your deployed system. Here are our top **5** vulnerability assessment observations:

1. We have assessed **16** of your deployed hosts, and identified **2263** unique installed IT Products.
With **42** candidate package/application/hardware/OS(s) that have **519** publicly known vulnerabilities.
2. Among all **519** publicly known vulnerabilities, there are **42** publicly available proof of concept exploit codes that can exploit some of the vulnerabilities.
3. Among all **519** publicly known vulnerabilities, **2** of them have public mitigation patches available.
4. Among all **519** publicly known vulnerabilities, **417** of them can be exploited via a NETWORK attack vector.
5. Among all **519** publicly known vulnerabilities, **90** of them can only be exploited via a LOCAL attack vector.



FIGURE 6.13. OpenStack Vulnerability Assessment Report's Executive Summary

tic facts that VULCAN tool generated after the vulnerability indexing (which includes semantic information about each identified IT product exposure to vulnerability-based threats

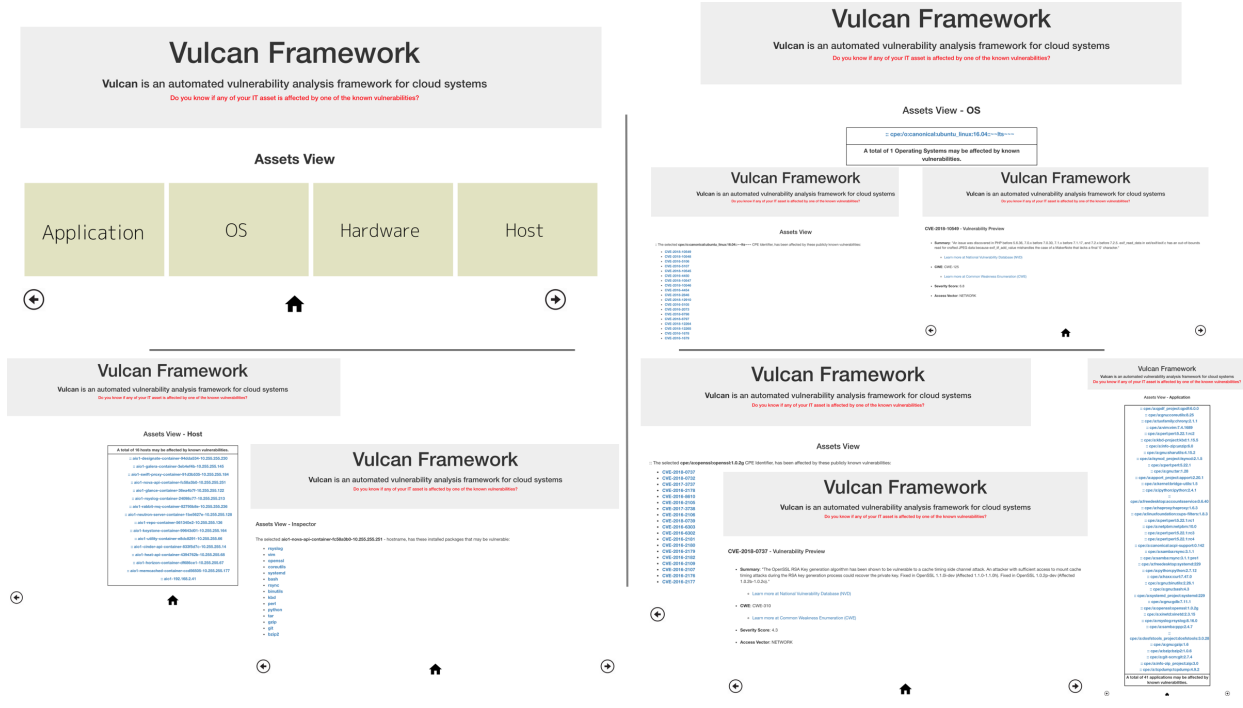


FIGURE 6.14. OpenStack Vulnerability Assessment Report's Asset View

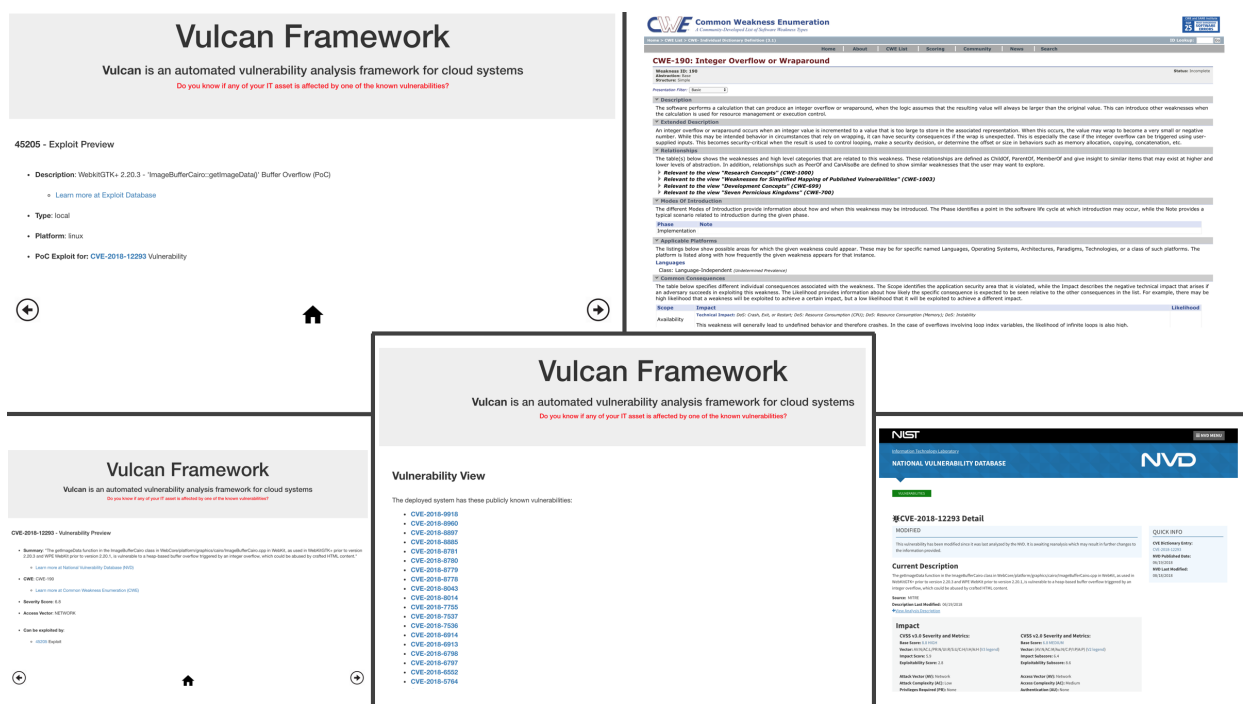


FIGURE 6.15. OpenStack Vulnerability Assessment Report's Vulnerability View

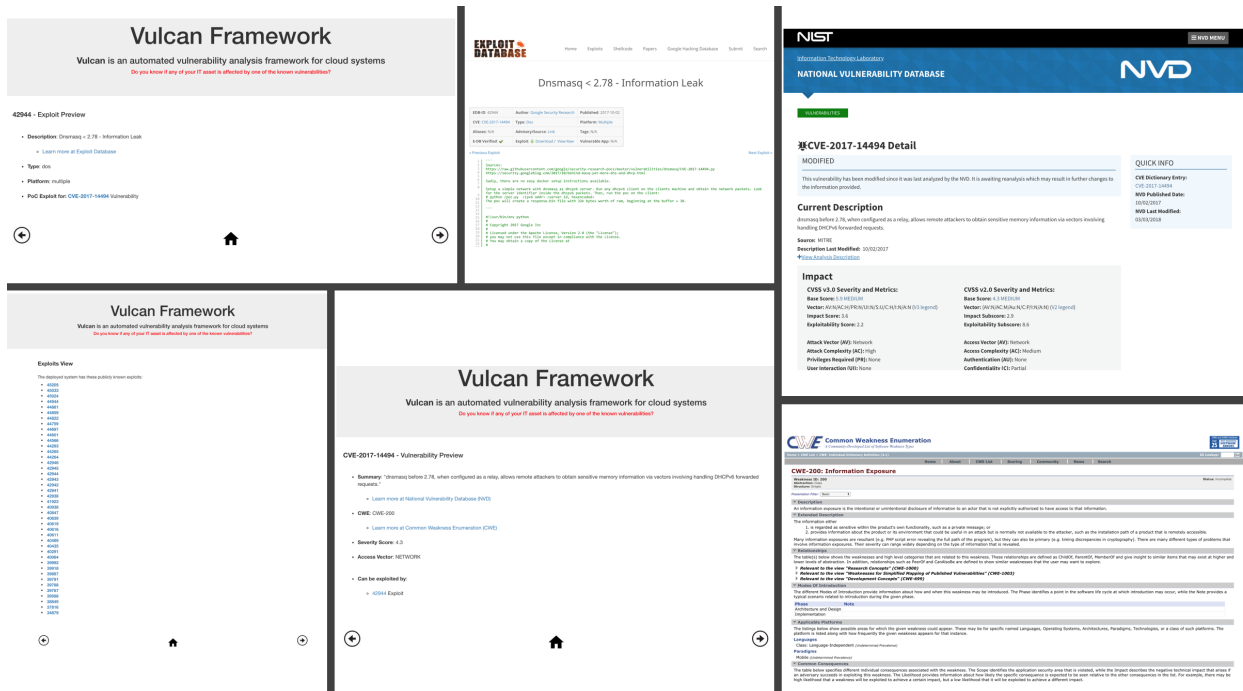


FIGURE 6.16. OpenStack Vulnerability Assessment Report's Exploit View

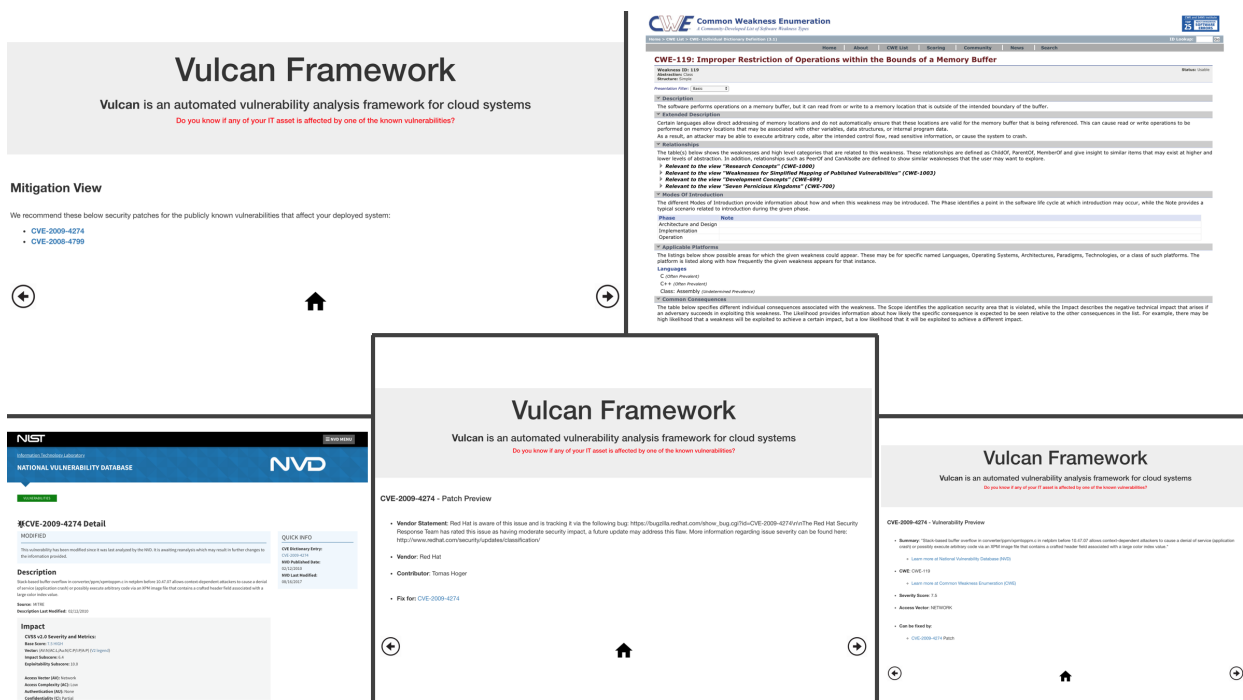


FIGURE 6.17. OpenStack Vulnerability Assessment Report's Mitigation View

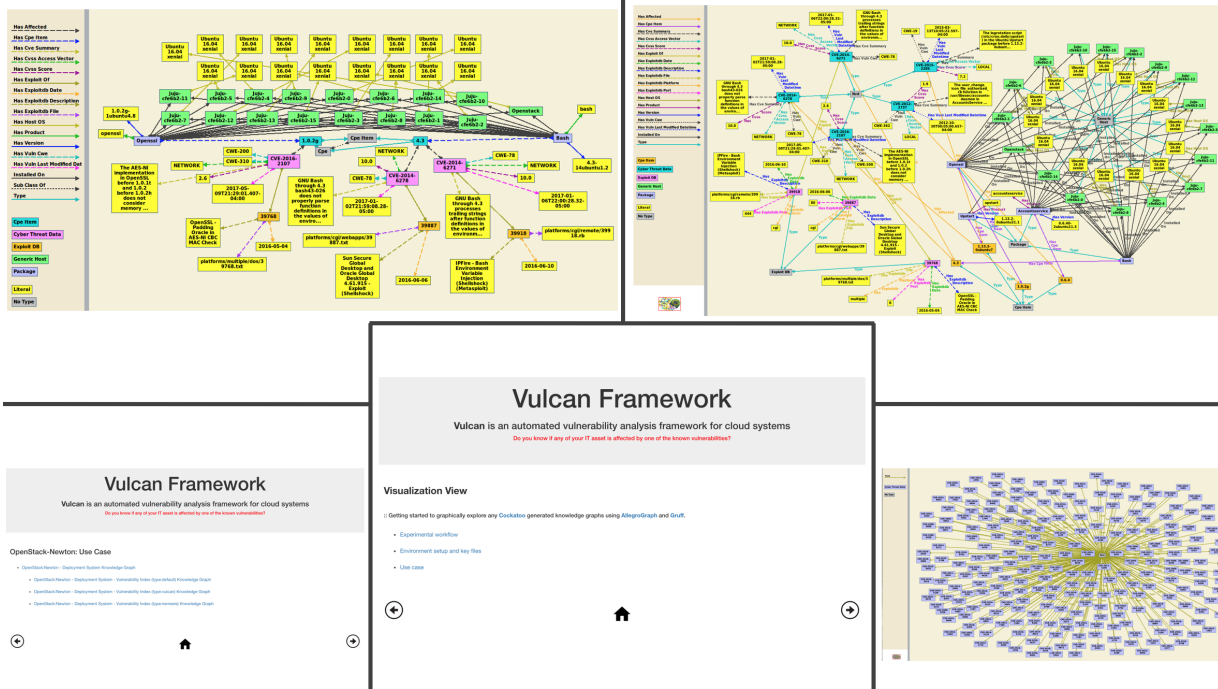


FIGURE 6.18. OpenStack Vulnerability Assessment Report's Visualization View

Vulnerability Index OKB	Triples
Ansible: OpenStack Queens	123,825
Bitnami: Django 1.11.14	13,846
Bitnami: Joomla 3.8.10	14,149
Bitnami: Drupal 8.5.5	14,153
Bitnami: Lamp 7.2.8	14,786
Bitnami: WordPress 4.9.7	14,474
Office 2016	2,431
OpenSSL 1.1.0a	1,263

TABLE 6.5. IT Systems Vulnerability Index OKBs – Generated by VULCAN

in terms of known vulnerabilities, exploits, and mitigation). Each of the generated vulnerability indexes can be used to render vulnerability assessment reports using the VULCAN tool's Report feature. In addition, a user can launch our HUMMING chatbot (See Fig-

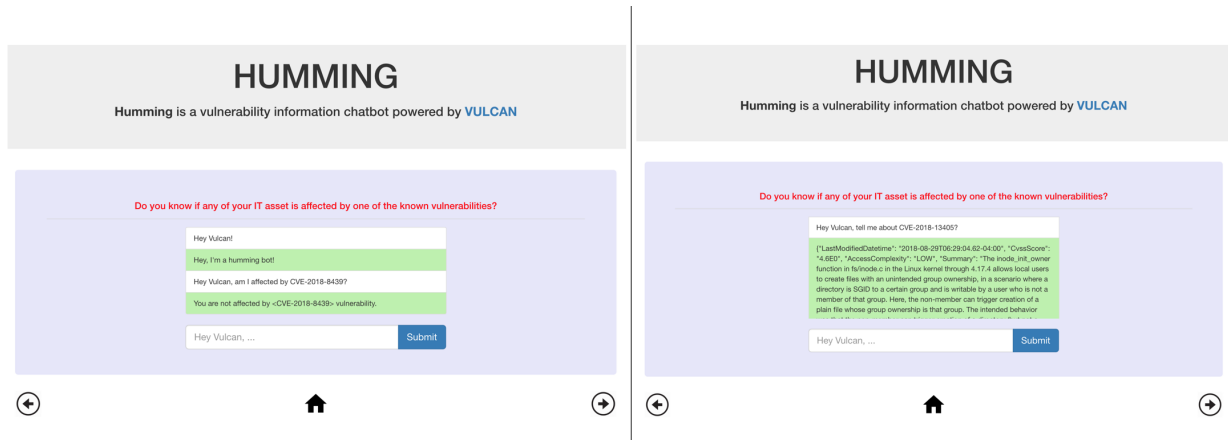


FIGURE 6.19. HUMMING – Dialogue Example

ure 6.10) for a dialogue based interaction (via the “*Query*” navigation) to ask questions about the security posture of their IT system. Figure 6.19 shows an example of such a dialogue between a user and our HUMMING chatbot while leveraging all VULCAN thus generated vulnerability indexes.

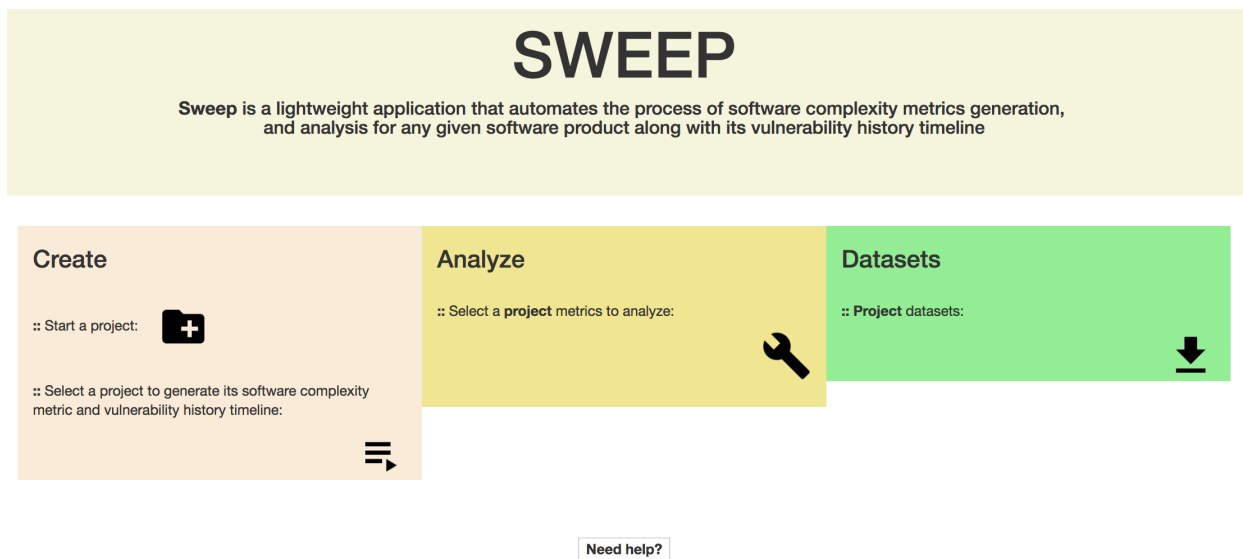


FIGURE 6.20. COCKATOO – SWEEP Tool

For individual software products such as *OpenSSL* [40], our COCKATOO tools SWEEP (See Figure 6.20) and PREDICTION (See Figure 6.21) can be used to study a given product and provide a web service to predict the number of unknown vulnerabilities that are likely to be discovered in that product’s new release. An example is illustrated in Figure 6.22 for

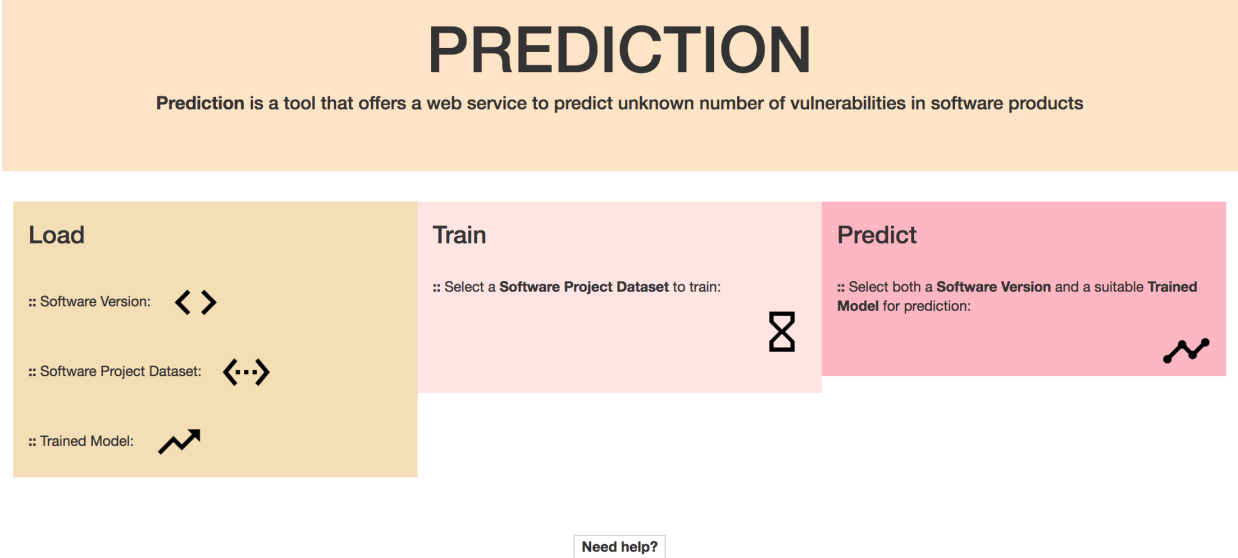


FIGURE 6.21. COCKATOO – PREDICTION Tool

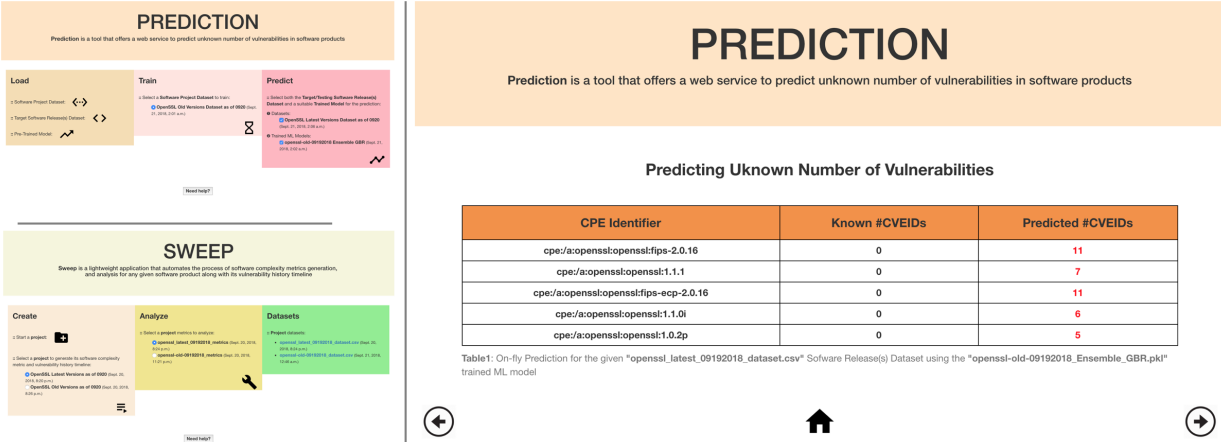


FIGURE 6.22. OpenSSL – Predictive Experiment Example

the latest OpenSSL product’s releases. These predictive findings can be used within our NEMESIS tool to update the risk score estimation for a given IT system (e.g., *OpenSSL 1.1.1*).

These vulnerability index OKBs (illustrated in Figure 6.11) can then be loaded into NEMESIS tool for various risk assessment tasks.

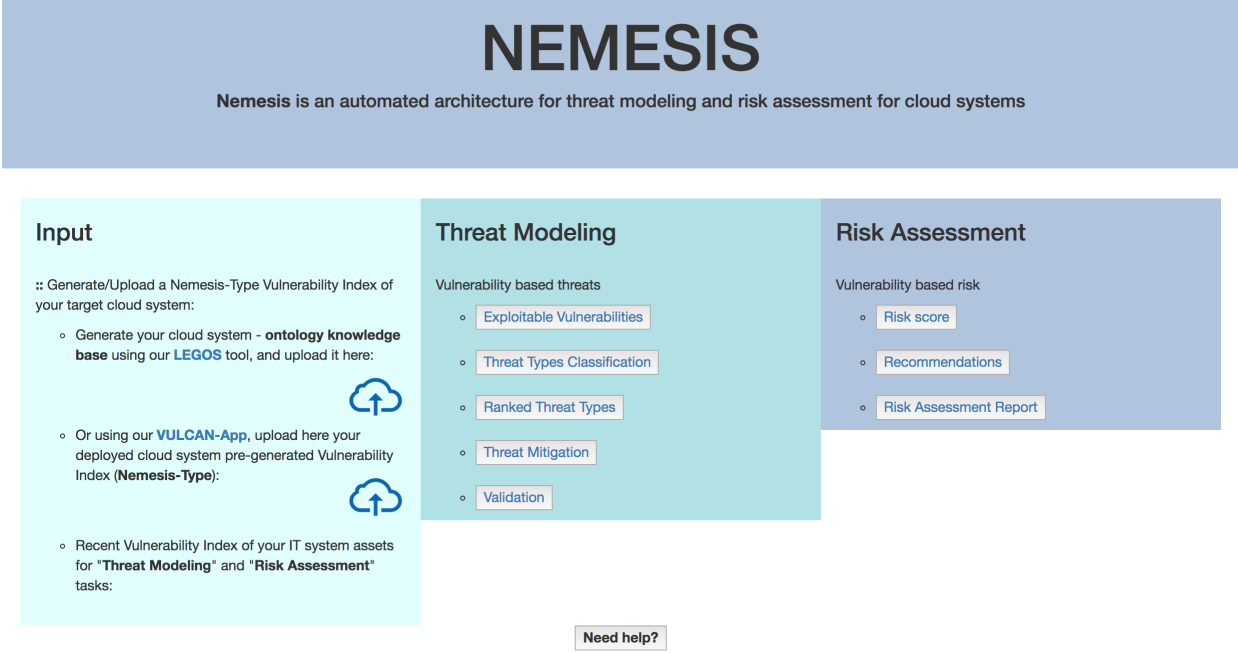


FIGURE 6.23. COCKATOO – NEMESIS Tool

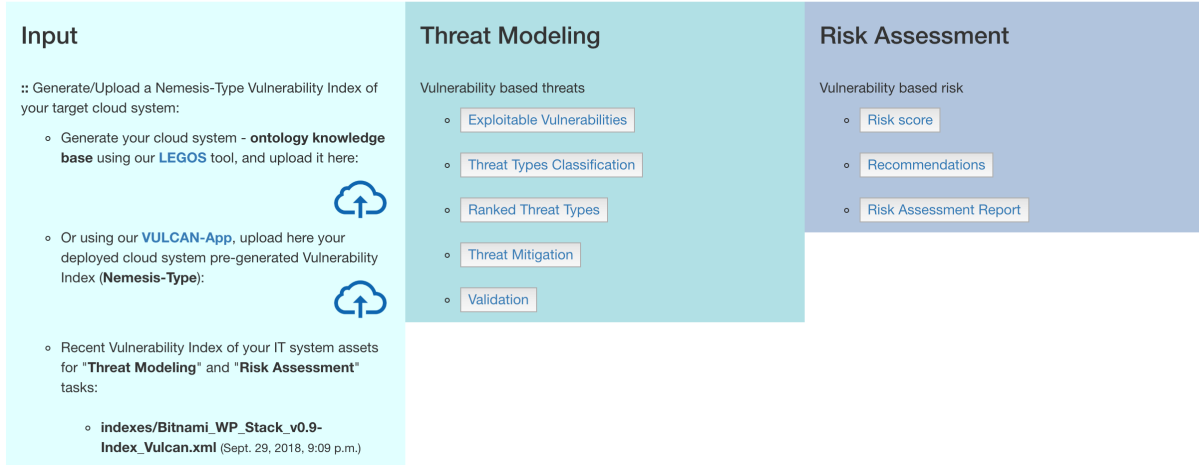
6.4.3. Risk Assessment

Within our COCKATOO system, we can use the NEMESIS tool (See Figure 6.23) to model vulnerability-based threats facing any given IT system such as ones shown in Table 6.2 via the “*Threat Modeling*” navigation. In addition, for each identified threat instance, we can use “*Risk Assessment*” feature of the NEMESIS tool to estimate its risk and provide recommendations to minimize the risk.

Let us assume that there is an organization that has one of their services running on a legacy system such as one of an old version of a WordPress stack shown in Table 6.3. As an example, we use configurations shown in Table 6.3 (which are compatible with our LEGOS supported IT assets data formats) to perform our NEMESIS assessments. As shown in Figure 6.24, NEMESIS will prompt the user to upload the IT system configurations (See Table 6.3) via a LEGOS interface or if the user has previously used our VULCAN tool, to upload VULCAN-generated generated vulnerability-index ontology knowledge base (OKB). Then NEMESIS will use the vulnerability-index OKB to begin the assessment. If the IT system configurations provided via LEGOS are the only inputs available for assessment,

NEMESIS

Nemesis is an automated architecture for threat modeling and risk assessment for cloud systems



Need help?

FIGURE 6.24. NEMESIS Portal with a User Input of their IT System Vulnerability Index OKB

Ranked Threat Types

Threat Type	#Instances
Denial of Service	134
Spoofting	79
Information Disclosure	96
Reputation	115
Elevation of Privilege	95

STRIDE threat types ranked based on their number of instances

Threat Type	Severity [0-10]
Denial of Service	5.83
Spoofting	5.4
Information Disclosure	5.34
Reputation	5.95
Elevation of Privilege	5.32

STRIDE threat types ranked based on the mean severity of their inherent vulnerabilities

STRIDE Threat Types - Classification

Vulnerability	Description	A	S	I	D	E	R
CVE-2018-1044	... (description)						
CVE-2018-1045	... (description)						
CVE-2018-1046	... (description)						
CVE-2018-1047	... (description)						
CVE-2018-1048	... (description)						
CVE-2018-1049	... (description)						
CVE-2018-1050	... (description)						
CVE-2018-1051	... (description)						
CVE-2018-1052	... (description)						
CVE-2018-1053	... (description)						
CVE-2018-1054	... (description)						
CVE-2018-1055	... (description)						
CVE-2018-1056	... (description)						
CVE-2018-1057	... (description)						
CVE-2018-1058	... (description)						
CVE-2018-1059	... (description)						
CVE-2018-1060	... (description)						

Denial of service -- Threat Type Validation

Affected Applications:

- cpe:/a:wordpress:wordpress:2.2
- cpe:/a:php:php:5.2.3
- cpe:/a:apache:apache:2.0.59

Other Affected Operating Systems:

- cpe:/o:redhat:enterprise_linux:6.0
- cpe:/o:canonical:ubuntu_linux:14.04--lts---
- cpe:/o:canonical:ubuntu_linux:14.04--lts---
- cpe:/o:canonical:ubuntu_linux:17.10
- cpe:/o:fedora:project:fedora:16
- cpe:/o:ubuntu:ubuntu:space:16.10
- cpe:/o:sugarmatic:os_x:10.6.0
- cpe:/o:canonical:ubuntu_linux:16.04--lts---
- cpe:/o:fedora:project:fedora:17
- cpe:/o:openbsd:openbsd:4.8
- cpe:/o:debian:debian_linux:8.0
- cpe:/o:redhat:enterprise_linux:7.0
- cpe:/o:oracle:solaris:10
- cpe:/o:google:android
- cpe:/o:fedora:project:fedora:18
- cpe:/o:metasploit:metasploit:5.1
- cpe:/o:debian:debian_linux:7.0

Exploitable Vulnerabilities

Vulnerability	Description
CVE-2006-5855	... (description)
CVE-2007-4507	... (description)
CVE-2009-2324	... (description)
CVE-2011-4865	... (description)
CVE-2011-3398	... (description)
CVE-2010-2026	... (description)
CVE-2011-1692	... (description)
CVE-2007-3760	... (description)
CVE-2011-1934	... (description)
CVE-2011-0708	... (description)

Input

- Generate/Upload a Nemesis-Type Vulnerability Index of your target cloud system:
 - Generate your cloud system - **ontology knowledge base** using our **LEGOS** tool, and upload it here:
 - Or using our **VULCAN-App**, upload here your deployed cloud system pre-generated Vulnerability Index (**Nemesis-Type**):
 - Recent Vulnerability Index of your IT system assets for **"Threat Modeling"** and **"Risk Assessment"** tasks:
 - indexes/Bitnami_WP_Stack_v0.9-Index_Vulcan.xml (Sept. 29, 2018, 9:09 p.m.)

Threat Modeling

Vulnerability based threats

- Exploitable Vulnerabilities
- Threat Types Classification
- Ranked Threat Types
- Threat Mitigation
- Validation

Risk Assessment

Vulnerability based risk

- Risk score
- Recommendations
- Risk Assessment Report

Recommended Mitigation Techniques:

- Authentication
- Authorization
- Filtering
- Throttling
- Quality of service

FIGURE 6.25. NEMESIS Threat Modeling – Key Findings for a WordPress Stack

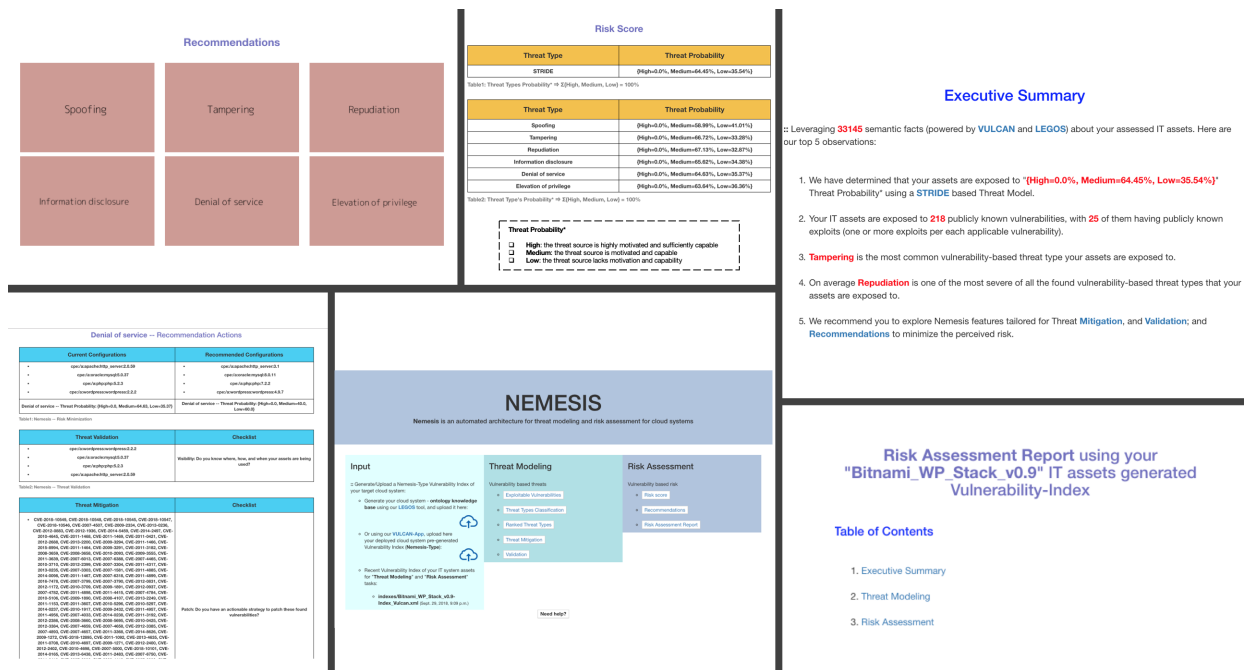


FIGURE 6.26. NEMESIS Risk Assessment – Key Findings for a WordPress Stack

then NEMESIS will initiate the vulnerability-index OKB generation using VULCAN API. Figure 6.25 illustrates our NEMESIS automated threat modeling approach for the given IT system (See Table 6.3) and shows key findings along assessment categories such as *Exploitable Vulnerabilities*, *Threat Types Classification*, *Ranked Threat Types*, *Threat Mitigation*, and *Validation*. Figure 6.26 shows NEMESIS’s risk assessment findings in terms of the risk score that each modeled threat instances may expose the organization to. In addition, our NEMESIS tool provides recommendations for each threat type category, for example “Denial of Service” threat type with all its found 134 threat instances (See Figure 6.25), in terms of what a user can do to minimize the estimated risk (e.g., patching the found vulnerabilities or upgrading old IT products to newer versions).

The NEMESIS tool can evaluate any vulnerability-based threat that affects a large IT system deployment such as ones shown in Table 6.2. For example, Figure 6.27 illustrates some key findings from NEMESIS’s evaluation of all found vulnerability-based threats that an example OpenStack deployment (See Figure 3.9) is exposed to, and its risk exposure.

(on an evolving basis) security experts knowledge and experiences into our ontology models and orchestration frameworks. These ontology models when populated with relevant data sources turns into semantically rich ontology knowledge bases (OKBs). These OKBs in turns power our orchestration frameworks into a reliable and automated web application and service offerings. These on-demand applications and services within the COCKATOO system enable any user (e.g., a novice to an experienced security professional) to perform threat assessment and mitigation for their IT systems, large or small.

The COCKATOO system currently has some limitations such as performance overhead of some tools, limited data-feed sources for our evolving knowledge graph, and reliance on some third-party commercial software products. For instance, some performance overheads are due to third-party software that we rely on to maintain our ontology models and knowledge bases and to automate software complexity metrics generation. Another performance overhead that is a result of our COCKATOO system codebase can be minimized by optimizing the code with more efficient programming languages, as well as, parallelizing tasks. The COCKATOO system already employs a number of performance tuning techniques such as caching for our web stack, indexing for various semantic graph database processing tasks, reliable and adaptable technology stack (See Figure 6.4). The main sources of cyber threat data-feeds enable us to address important questions that revolve around vulnerability-based threats to IT system. We plan to include a number of additional sources to extend our ontology models so that we can represent information about IT systems' misconfiguration and misuse of security implications. Since most security attacks rely on a number of attack vectors for successful security breaches, we plan to empower users with a holistic solution. Also, we plan to complete the adoption of open-source technology solutions so that our COCKATOO system can be freely available to all of our users that we had in mind when designing it.

In this Chapter, we have presented our approach to how the COCKATOO system is designed and implemented as a holistic suite of security tools. We have evaluated each of COCKATOO tools using a sample of IT systems that we have studied and deployed.

Within the COCKATOO system, we are providing a robust solution that has the potential to help individuals and organizations to stay ahead of any vulnerability-based threat that their IT systems may be exposed to. The COCKATOO system related materials such as tools' codebase, white papers, and tools video demos are maintained within the University of North Texas, Computer Science and Engineering – Computer Computer Systems Research Lab (CSRL)" [26], and under the CSRL – COCKATOO project page [25].

In the next Chapter 7, we include concluding thoughts and plans for future expansions.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this dissertation, I have presented our research on the vulnerability-based threats facing IT systems (e.g., cloud system deployment). Our research can help individuals and organizations using IT systems stay ahead of known and unknown vulnerability-based threats that may have a damaging impact on their core mission goals.

I have presented our ontological approach using state-of-the-art ontology engineering techniques to build a cyber threat knowledge base, a working knowledge base of a user deployed IT system, and link these knowledge bases for automated threat assessment and mitigation tasks. Our cyber threat knowledge bases reflect publicly known information about vulnerability, vulnerability-exploits, and vulnerability-mitigation information that affects IT systems such as applications, hardware, and operating systems. The user-generated IT system knowledge base is used by our automated systems to gain visibility into all user's exploitable IT assets.

I have presented my significant contributions in terms of the VULCAN, NEMESIS, and COCKATOO systems. VULCAN captures all vulnerabilities, exploits, and mitigation information associated with IT systems. The VULCAN system enables vulnerability assessment of complex IT systems (e.g., cloud systems). I have presented our NEMESIS system that categorizes and ranks threat types exposed by the vulnerabilities that exist in an IT system. Also, NEMESIS uses a Bayesian model to quantify overall threat risks and recommends alternate system configurations to reduce the perceived IT system's risks. COCKATOO integrates VULCAN, NEMESIS and other tools (IKAWAFARM, LEGOS, HUMMING, SWEEP, and PREDICTION) into a holistic analysis environment. The COCKATOO system enables novices or experienced users to evaluate the security posture of their deployed IT systems.

With this research work, we have provided a new understanding and solutions to the problem of assessing and mitigating vulnerability-based threats that affect IT systems. Our goal is to encourage new ideas that security researchers and practitioners can use to develop

new solutions to help individuals and organizations protect their IT systems against targeted threats.

We plan to advance the development of ontology-based solutions that are suitable for industry setting and enabling end-users to have controls in assessing and taking a course of action in regards to their IT systems security posture.

In Table 7.1, we present a summary of each of COCKATOO's tools' future extensions. For instance,

- With the LEGOS tool, we plan to extend our IT system ontology model to represent a wide variety of security-related telemetry data from any computing device, including configuration information so that vulnerability prone configurations can be identified. Also, LEGOS can collect network and computer's resources utilization towards building a baseline for normal behaviors and enable other COCKATOO tools to detect abnormal behaviors and correlate them with the assessed security posture of an IT system regarding found vulnerabilities and threat types exposure.
- With the VULCAN tool, we plan to develop tools to better visualize vulnerabilities and their relationships using 3D rendering of our ontology graphs (for example, using Virtual Reality technologies). We will develop Virtual Reality applications that enable exploration of the generated VULCAN ontology knowledge base that captured a given user's IT system vulnerability status, including information about found vulnerabilities.
- With the SWEEP tool, we plan to explore computing software complexity measures from binary sources of software products when source code is not available. Using the PREDICTION tool, we will focus on exploring security-specific software complexity measures to gain insights into programs that could have hidden vulnerabilities (that are yet to be discovered), or that could have malicious codes (embedded by motivated threat actors).

COCKATOO – Tool	Future Extension
IKAWAFARM	Extend our cyber threat data ontology to capture additional data sources and feeds (such as IoT and hardware vulnerability databases). Incorporate STIX [88] in the IKAWAFARM toolkit to capture cyber threat intelligence feeds from industry, community, and open-source (e.g., social media sites, blogs, etc.) feeds. Expose a semantic API to our COCKATOO tools to enable basic to complex queries on the newly collected and represented threat data feeds.
LEGOS	Extend our IT system ontology model to represent a wide variety of security-related telemetry data from any computing device, including configuration information so that vulnerability prone configurations can be identified.
VULCAN	Extend VULCAN to track IT systems through their lifetime. Develop tools to better visualize vulnerabilities and their relationships using 3D rendering of our ontology graphs (for example, using Virtual Reality technologies).
HUMMING	Develop more semantically meaningful chatbot dialogues relying on the feedback from several conducted study cases.
NEMESIS	Explore context-dependent ways of assigning severity scores to vulnerabilities. Use semantic word disambiguation and NLP to better classify vulnerabilities into threat types.
SWEEP	Explore computing software complexity measures from binary sources (when source code is not available). Explore security-specific software complexity measures.
PREDICTION	Further validate our prediction models for a wide variety of software products.

TABLE 7.1. COCKATOO Tools – Future Extensions

REFERENCES

- [1] Jans Aasman, *Allegro graph: Rdf triple database*, Cidade: Oakland Franz Incorporated (2006).
- [2] Stokes Adam and McCracken Mike, *Installing conjure-up*, <https://docs.conjure-up.io/stable/en/user-manual#installing-conjure-up>, October 2018.
- [3] Danial Al, *Count lines of code*, <https://github.com/AlDanial/cloc>, October 2018.
- [4] Omar H Alhazmi and Yashwant K Malaiya, *Modeling the vulnerability discovery process*, 16th IEEE International Symposium on Software Reliability Engineering, 2005. ISSRE 2005., IEEE, 2005, pp. 10–pp (english).
- [5] ———, *Prediction capabilities of vulnerability discovery models*, Reliability and Maintainability Symposium, 2006. RAMS'06. Annual, IEEE, 2006, pp. 86–91 (english).
- [6] Omar H Alhazmi, Yashwant K Malaiya, and Indrajit Ray, *Measuring, analyzing and predicting security vulnerabilities in software systems*, Computers & Security, vol. 26, Elsevier, 2007, pp. 219–228 (english).
- [7] Cloud Security Alliance, *The notorious nine - cloud computing top threats in 2013*, https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf, October 2018.
- [8] Amazon, *Amazon ec2 security groups*, <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html>, October 2018.
- [9] ———, *Amazon elastic compute cloud (amazon ec2)*, <https://aws.amazon.com/ec2/>, October 2018.
- [10] ———, *Amazon machine images (ami)*, <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html>, October 2018.
- [11] ———, *Amazon web services*, <http://aws.amazon.com>, October 2018.
- [12] The Kubernetes Authors, *Kubernetes - production-grade container orchestration*, <https://kubernetes.io>, October 2018.

- [13] Microsoft Azure, *Microsoft azure machine learning*, <https://studio.azureml.net>, October 2018.
- [14] I. Bedini and B. Nguyen, *Automatic ontology generation: State of the art*, PRiSM Laboratory Technical Report. University of Versailles (2007).
- [15] Akhil Behl and Kanika Behl, *An analysis of cloud computing security issues*, Information and Communication Technologies (WICT), 2012 World Congress on, IEEE, 2012, pp. 109–114.
- [16] Mehran Bozorgi, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker, *Beyond heuristics: learning to classify vulnerabilities and predict exploits*, Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2010, pp. 105–114 (english).
- [17] Lee Chen-Yu, Kamongi Patrick, Kavi Krishna, and Gomathisankaran Mahadevan, *Optimus: A framework of vulnerabilities, attacks, defenses and sla ontologies*, International Journal of Next-Generation Computing 6 (2016), no. 1.
- [18] Istehad Chowdhury and Mohammad Zulkernine, *Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities*, Journal of Systems Architecture, vol. 57, Elsevier, 2011, pp. 294–313 (english).
- [19] IBM Cloud, *What is cloud computing?*, <https://www.ibm.com/cloud/learn/what-is-cloud-computing>, October 2018.
- [20] Ubuntu Community, *Apachemysqlphp*, <https://help.ubuntu.com/community/ApacheMySQLPHP>, October 2018.
- [21] Bootstrap core team and contributors, *Bootstrap*, <https://getbootstrap.com>, October 2018.
- [22] Microsoft Corporation, *Identifying techniques that mitigate threats*, [http://msdn.microsoft.com/en-us/library/ee798428\(v=cs.20\).aspx](http://msdn.microsoft.com/en-us/library/ee798428(v=cs.20).aspx), October 2018.
- [23] ———, *Microsoft eop game*, <https://www.microsoft.com/en-us/SDL/adopt/eop.aspx>, October 2018.
- [24] CSA, *Cloud security alliance*, <https://cloudsecurityalliance.org>, October 2018.

- [25] CSRL, *Cockatoo toolchain*, <https://csrl.cse.unt.edu/content/cockatoo-toolchain>, October 2018.
- [26] ———, *Computer systems research lab*, <https://csrl.cse.unt.edu/>, October 2018.
- [27] DarkReading, *Bank ddos attacks resume: Wells fargo confirms disruptions*, <https://www.darkreading.com/attacks-and-breaches/bank-ddos-attacks-resume-wells-fargo-confirms-disruptions/d/d-id/1109271>, October 2018.
- [28] Aleksandar Donevski, Sasko Ristov, and Marjan Gusev, *Security assessment of virtual machines in open source clouds*, Information & Communication Technology Electronics & Microelectronics (MIPRO), 2013 36th International Convention on, IEEE, 2013, pp. 1094–1099.
- [29] Drupal, *Drupal software stack*, <https://www.drupal.org/>, October 2018.
- [30] Andreas Ekelhart, Stefan Fenz, Markus D Klemen, and Edgar R Weippl, *Security ontology: Simulating threats to corporate assets*, International Conference on Information Systems Security, Springer, 2006, pp. 249–259.
- [31] EsecurityPlanet, *Majority of u.s. banks were hit by ddos attacks in 2012*, <http://www.esecurityplanet.com/network-security/majority-of-u.s.-banks-were-hit-by-ddos-attacks-in-2012.html>, October 2018.
- [32] Katheryn A Farris, Ankit Shah, George Cybenko, Rajesh Ganesan, and Sushil Jajodia, *Vulcon: A system for vulnerability prioritization, mitigation, and management*, ACM Transactions on Privacy and Security (TOPS) 21 (2018), no. 4, 16.
- [33] Stefan Fenz, *Ontology-based generation of IT-security metrics*, Proceedings of the 2010 ACM Symposium on Applied Computing, ACM, 1 2010, pp. 1833–1839.
- [34] Stefan Fenz, *An ontology-and bayesian-based approach for determining threat probabilities*, Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ACM, 2011, pp. 344–354.
- [35] ———, *An ontology-based approach for constructing bayesian networks*, Data & Knowledge Engineering, vol. 73, Elsevier, 2012, pp. 73–88.

- [36] FIRST, *Common vulnerability scoring system*, <http://www.first.org/cvss>, October 2018.
- [37] Stanford Center for Biomedical Informatics Research, *Protege editor*, <http://protege.stanford.edu>, October 2018.
- [38] Django Software Foundation, *Django*, <https://www.djangoproject.com>, October 2018.
- [39] Electronic Frontier Foundation, *Assessing your risks*, <https://ssd.eff.org/en/module/assessing-your-risks>, October 2018.
- [40] OpenSSL Software Foundation, *Openssl*, <https://www.openssl.org>, October 2018.
- [41] Openstack Foundation, *Devstack*, <http://devstack.org>, October 2018.
- [42] ———, *Openstack releases*, <https://wiki.openstack.org/wiki/Releases>, October 2018.
- [43] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang, *Hermit: an owl 2 reasoner*, *Journal of Automated Reasoning* 53 (2014), no. 3, 245–269.
- [44] GNU, *Gnu general public license*, <https://www.gnu.org/licenses/gpl-3.0.en.html>, October 2018.
- [45] Asuncion Gomez-Perez, Mariano Fernández-López, and Oscar Corcho, *Ontological engineering: with examples from the areas of knowledge management, e-commerce and the semantic web*, Springer Science & Business Media, 2006.
- [46] Todd L Graves, Alan F Karr, James S Marron, and Harvey Siy, *Predicting fault incidence using software change history*, *IEEE Transactions on Software Engineering*, vol. 26, IEEE, 2000, pp. 653–661 (english).
- [47] Top Threats Working Group, *The notorious nine: Cloud computing top threats in 2013*, Tech. report, Cloud Security Alliance, September 2016.
- [48] ———, *Top threats cloud computing survey 2012*, Tech. report, Cloud Security Alliance, September 2016.
- [49] ———, *The treacherous 12 - cloud computing top threats in 2016*, Tech. report, Cloud Security Alliance, September 2016.

- [50] Thomas R Gruber et al., *A translation approach to portable ontology specifications*, Knowledge acquisition 5 (1993), no. 2, 199–220.
- [51] Nils Gruschka and Meiko Jensen, *Attack surfaces: A taxonomy for attacks on cloud services.*, IEEE CLOUD, 2010, pp. 276–279.
- [52] M. Guo and J.A. Wang, *An ontology-based approach to model common vulnerabilities and exposures in information security*, ASEE Southeast Section Conference, 2009.
- [53] Michael Iannacone, Shawn Bohn, Grant Nakamura, John Gerth, Kelly Huffer, Robert Bridges, Erik Ferragut, and John Goodall, *Developing an ontology for cyber security knowledge graphs*, Proceedings of the 10th Annual Cyber and Information Security Research Conference, ACM, 2015, p. 12.
- [54] ICASI, *Common vulnerability reporting framework (cvrf)*, <http://www.icaso.org/cvrf>, October 2018.
- [55] Adobe Systems Inc., *Adobe – security bulletins and advisories*, <https://helpx.adobe.com/security.html>, October 2018.
- [56] Apple Inc., *Apple security updates*, <https://support.apple.com/en-us/HT201222>, October 2018.
- [57] Equifax Inc., *2017 cybersecurity incident & important consumer information*, <https://www.equifaxsecurity2017.com/consumer-notice>, October 2018.
- [58] Franz Inc., *Agwebview web browser server for allegrograph triple stores*, <http://franz.com/agraph/agwebview>, October 2018.
- [59] ———, *Allegrograph*, <http://franz.com/agraph/allegrograph>, October 2018.
- [60] ———, *Gruff: A grapher-based triple-store browser for allegrograph*, <https://franz.com/agraph/gruff>, October 2018.
- [61] Microsoft Inc., *Boosted decision tree regression*, <https://msdn.microsoft.com/en-us/library/azure/dn905801.aspx>, October 2018.
- [62] ———, *The stride threat model*, [https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878\(v=cs.20\)](https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)), October 2018.
- [63] Oracle Inc., *Virtualbox*, <https://www.virtualbox.org>, October 2018.

- [64] Scientific Toolworks Inc., *Understand scitools*, <https://scitools.com>, October 2018.
- [65] Cybersecurity journal, *Cyber matters: It's time to decide what kind of risk is acceptable*, <https://www.cybersecurityjournal.org/cyber-matters-its-time-to-decide-what-kind-of-risk-is-acceptable/>, October 2018.
- [66] Scikit learn developers, *Gradient boosting for regression*, <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor>, October 2018.
- [67] Chen-Yu Lee, Krishna M. Kavi, Mahadevan Gomathisankaran, and Patrick Kamongi, *Security Through Software Rejuvenation*, The Ninth International Conference on Software Engineering Advances (ICSEA), IARIA, 2014, pp. 347–353 (english).
- [68] Canonical Ltd, *The lxd container hypervisor*, <https://www.ubuntu.com/containers/lxd>, October 2018.
- [69] ———, *Scale out with ubuntu server*, <https://www.ubuntu.com/server>, October 2018.
- [70] Pratyusa K Manadhata and Jeannette M Wing, *An attack surface metric*, IEEE Transactions on Software Engineering 37 (2011), no. 3, 371–386.
- [71] McAfee, *Threat landscape dashboard*, <https://www.mcafee.com/threat-center/threat-landscape-dashboard/>, October 2018.
- [72] Peter Mell and Timothy Grance, *The nist definition of cloud computing*, Tech. Report 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.
- [73] P. Meunier, *Classes of vulnerabilities and attacks*, Wiley Handbook of Science and Technology for Homeland Security (2008).
- [74] Microsoft, *Microsoft office*, <https://www.office.com/>, October 2018.
- [75] MITRE, *Common attack pattern enumeration and classification (capec)*, <http://capec.mitre.org>, October 2018.

- [76] ———, *Common vulnerabilities and exposures (cve)*, <http://cve.mitre.org>, October 2018.
- [77] ———, *Common weakness enumeration (cwe)*, <http://cwe.mitre.org>, October 2018.
- [78] ———, *Cyber observable expression (cybox)*, <https://cybox.mitre.org>, October 2018.
- [79] ———, *Malware attribute enumeration and characterization (maec)*, <https://maec.mitre.org>, October 2018.
- [80] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Avi Patel, and Muttukrishnan Rajarajan, *A survey on security issues and solutions at different layers of cloud computing*, *The Journal of Supercomputing* 63 (2013), no. 2, 561–592.
- [81] Mark A Musen, *The protégé project: a look back and a look forward*, *AI matters* 1 (2015), no. 4, 4–12.
- [82] NIST, *Common configuration enumeration (cce)*, <https://nvd.nist.gov/cce/index.cfm>, October 2018.
- [83] ———, *Common platform enumeration (cpe)*, <https://nvd.nist.gov/cpe.cfm>, October 2018.
- [84] ———, *Cve-2014-0160 detail*, <https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160>, October 2018.
- [85] ———, *National vulnerability database (nvd)*, <https://nvd.nist.gov>, October 2018.
- [86] ———, *Security content automation protocol*, <https://csrc.nist.gov/projects/security-content-automation-protocol/>, October 2018.
- [87] S Noel, E Harley, KH Tam, M Limiero, and M Share, *Cygraph: graph-based analytics and visualization for cybersecurity*, *Handbook of Statistics*, vol. 35, Elsevier, 2016, pp. 117–167.
- [88] OASIS, *Structured threat information expression (stix)*, <https://stix.mitre.org>, October 2018.

- [89] OWASP, *Threat risk modeling*, https://update-wiki.owasp.org/index.php/Threat_Risk_Modeling#DREAD, October 2018.
- [90] Kamongi Patrick, Kavi Krishna, and Gomathisankaran Mahadevan, *Predicting unknown vulnerabilities using software metrics and maturity models*, The Eleventh International Conference on Software Engineering Advances (ICSEA 2016), IARIA, August 2016.
- [91] Kamongi Patrick, Gomathisankaran Mahadevan, and Krishna Kavi, *Nemesis: Automated architecture for threat modeling and risk assessment for cloud computing*, The Sixth ASE International Conference on Privacy, Security, Risk and Trust (PASSAT), ASE, December 2014.
- [92] Kamongi Patrick, Kotikela Srujan, Kavi Krishna, Gomathisankaran Mahadevan, and Singhal Anoop, *Vulcan: Vulnerability assessment framework for cloud computing*, Proceedings of The Seventh International Conference on Software Security and Reliability. SERE (SSIRI) 2013, IEEE, June 2013, pp. 218–226.
- [93] Kamongi Patrick, Kotikela Srujan, Gomathisankaran Mahadevan, and Kavi Krishna, *A methodology for ranking cloud system vulnerabilities*, Proceedings of The Fourth International Conference on Computing, Communication and Networking Technologies (ICCCNT'13), 2013.
- [94] OpenStack project, *Newton*, <https://releases.openstack.org/newton>, October 2018.
- [95] ———, *Openstack*, <https://www.openstack.org>, October 2018.
- [96] ———, *Openstack core services*, <https://www.openstack.org/software/project-navigator>, October 2018.
- [97] ———, *Quickstart: Aio*, <https://docs.openstack.org/openstack-ansible/latest/user/aio/quickstart.html>, October 2018.
- [98] Rapid7, *Metasploit framework*, <https://metasploit.com>, October 2018.
- [99] Dark Reading, *Nsa-funded cauldron tool goes commercial*, <https://www>.

- darkreading.com/nsa-funded-cauldron-tool-goes-commercial/d/d-id/1131178, October 2018.
- [100] W3C Recommendation, *Owl 2 web ontology language: Document overview*, Tech. report, W3C, December 2012.
- [101] ———, *Sparql 1.1 query language*, Tech. report, W3C, March 2013.
- [102] ———, *Rdf 1.1 concepts and abstract syntax*, Tech. report, W3C, February 2014.
- [103] ———, *Rdf schema 1.1*, Tech. report, W3C, February 2014.
- [104] Offensive Security, *Offensive security's exploit database archive*, <https://www.exploit-db.com>, October 2018.
- [105] Tenable Network Security, *Nessus vulnerability scanner*, <http://www.tenable.com/products/nessus>, October 2018.
- [106] Amartya Sen and Sanjay Madria, *Off-line risk assessment of cloud service provider*, International Workshop on Cloud Security Auditing (CSA 2014), IEEE, 2014.
- [107] Dimitrios Settas, Antonio Cerone, and Stefan Fenz, *Enhancing ontology-based antipattern detection using bayesian networks*, Expert Systems with Applications, vol. 39, 8 2012, pp. 9041–9053.
- [108] O. Sheyner and J. Wing, *Tools for generating and analyzing attack graphs*, Formal methods for components and objects, Springer, 2004, pp. 344–371.
- [109] Oleg M Sheyner, *Scenario graphs and attack graphs*, Tech. report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2004.
- [110] Yonghee Shin, *Exploring complexity metrics as indicators of software vulnerability*, Proceedings of the 3rd International Doctoral Symposium on Empirical Software Engineering, Kaiserslautern, Germany, 2008, URL: http://www4.ncsu.edu/~yshin2/papers/esem2008ds_shin.pdf (english).
- [111] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A Osborne, *Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities*, IEEE Transactions on Software Engineering, vol. 37, IEEE, 2011, pp. 772–787 (english).

- [112] Yonghee Shin and Laurie Williams, *An empirical model to predict security vulnerabilities using code complexity metrics*, Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, ACM, 2008, pp. 315–317 (english).
- [113] ———, *Is complexity really the enemy of software security?*, Proceedings of the 4th ACM workshop on Quality of protection, ACM, 2008, pp. 47–50 (english).
- [114] Miguel-Angel Sicilia, Elena García-Barriocanal, Javier Bermejo-Higuera, and Salvador Sánchez-Alonso, *What are information security ontologies useful for?*, Research Conference on Metadata and Semantics Research, Springer, 2015, pp. 51–61.
- [115] A. Steele, *Ontological vulnerability assessment*, Web Information Systems Engineering—WISE 2008 Workshops, Springer, 2008, pp. 24–35.
- [116] Zareen Syed, Ankur Padia, Tim Finin, M Lisa Mathews, and Anupam Joshi, *Uco: A unified cybersecurity ontology.*, AAAI Workshop: Artificial Intelligence for Cyber Security, 2016.
- [117] Hassan Takabi, James B. D. Joshi, and Gail-Joon Ahn, *Security and privacy challenges in cloud computing environments*, IEEE Security and Privacy 8 (2010), no. 6, 24–31.
- [118] Takeshi Takahashi, Youki Kadobayashi, and Hiroyuki Fujiwara, *Ontological approach toward cybersecurity in cloud computing*, Proceedings of the 3rd international conference on Security of information and networks, ACM, 2010, pp. 100–109.
- [119] A Min Tjoa and Stefan Fenz, *Ontology- and bayesian-based threat probability determination*, Proceedings of the Junior Scientist Conference 2008, Vienna University of Technology, 11 2008, pp. 69–70.
- [120] J.A. Wang and M. Guo, *Ovm: an ontology for vulnerability management*, Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies, ACM, 2009, p. 34.
- [121] Ju An Wang, Minzhe Guo, Hao Wang, Min Xia, and Linfeng Zhou, *Ontology-based security assessment for software products*, Proceedings of the 5th Annual Workshop on

Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies, ACM, 2009, p. 15.

- [122] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia, *An attack graph-based probabilistic security metric*, IFIP Annual Conference on Data and Applications Security and Privacy, Springer, 2008, pp. 283–296.
- [123] Article Wikipedia, *Weighted arithmetic mean*, <https://stat.ethz.ch/R-manual/R-devel/library/stats/html/weighted.mean.html>, October 2018.
- [124] WordPress, *Wordpress software stack*, <https://wordpress.org/>, October 2018.
- [125] WSJ, *Big u.s. banks face increase in attempted cyberattacks*, <https://www.wsj.com/articles/big-u-s-banks-face-increase-in-attempted-cyberattacks-1538317920>, October 2018.
- [126] Su Zhang, Doina Caragea, and Xinming Ou, *An empirical study on using the national vulnerability database to predict software vulnerabilities*, Database and Expert Systems Applications, Springer, 2011, pp. 217–231 (english).