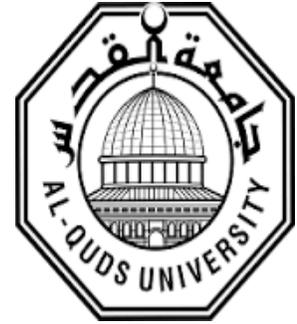


**Deanship of Graduate Studies
Al-Quds University**



Smart Intrusion Detection System for DMZ

“Mohamed Rami” Zuher Hassan Isifan

M.Sc. Thesis

Jerusalem – Palestine

1437 / 2016

Smart Intrusion Detection System for DMZ

Prepared by:

“Mohamed Rami” Zuher Hassan Isifan

B.Sc. Degree in Computer Science from An-Najah national university Palestine

Supervisor: Dr. Jad Najjar

Co. Supervisor: Dr. Rashid Jayosi

A thesis Submitted in Partial fulfillment of requirements for the degree of Master of Computer Science/Al-Quds University

1437 / 2016

Al-Quds University
Deanship of Graduate Studies
Computer Science Department



Thesis Approval

Smart Intrusion Detection System for DMZ

Prepared By : "Mohamed Rami" Zuher Hassan Isifan
Registration No: 21211780

Supervisor: Dr. Jad Najjar
Co. Supervisor: Dr. Rashid Jayosi

Master thesis submitted and accepted. Date: 4/6/2016

The names and signatures of the examining committee members are as follows:

- | | |
|--|---|
| 1. Head of Committee: Dr. Jad Najjar | Signature:  |
| 2. Committee Member: Dr. Rashid Jayosi | Signature:  |
| 3. Internal Examiner : Dr. Badie Saratwi | Signature:  |
| 4. External Examiner : Dr. Ahmad Alsadeh | Signature:  |

Jerusalem – Palestine
1437 / 2016

Dedication:

I dedicate this to my mother; for supporting me throughout my life.

“Mohamed Rami” Zuher Hassan Isifan.

Declaration

I certify that this thesis submitted for the degree of master, is the result of my own research, except where otherwise acknowledged, and that this study (or any part of the same) has not been submitted for a higher degree to any other university or institution.

Signed: *Mohamed*.....

“Mohamed Rami” Zuher Hassan Isifan

Date: 4/6/2016

Acknowledgements

I would like to express my sincere gratitude to those who gave me the assistance and support during my master study especially my mother.

I would like to thank professors, Dr. Badie Saratwi and Dr. Ahmad Alsadeh, who served on my thesis committee. Their comments and suggestions were invaluable. My deepest gratitude and appreciation goes to my supervisors Dr. Jad Najjar and Dr. Rashid Jayosi for their continuous supports and advices at all stages of my work.

Another word of special thanks goes to Al-Quds University, especially for all those in the Faculty of Graduate Studies / Computer Science department.

Abstract

Prediction of network attacks and machine understandable security vulnerabilities are complex tasks for current available Intrusion Detection System [IDS]. IDS software is important for an enterprise network. It logs security information occurred in the network. In addition, IDSs are useful in recognizing malicious hack attempts, and protecting it without the need for change to client's software. Several researches in the field of machine learning have been applied to make these IDSs better and smarter.

In our work, we propose approach for making IDSs more analytical, using semantic technology. We made a useful semantic connection between IDSs and National Vulnerability Databases [NVDs], to make the system semantically analyzed each attack logged, so it can perform prediction about incoming attacks or services that might be in danger. We built our ontology skeleton based on standard network security. Furthermore, we added useful classes and relations that are specific for DMZ network services. In addition, we made an option to allow the user to update the ontology skeleton automatically according to the network needs.

Our work is evaluated and validated using four different methods: we presented a prototype that works over the web. Also, we applied KDDCup99 dataset to the prototype. Furthermore, we modeled our system using queuing model, and simulated it using Anylogic simulator. Validating the system using KDDCup99 benchmark shows good results low false positive attacks prediction. Modeling the system in a queuing model allows us to predict the behavior of the system in a multi-users system for heavy network traffic.

Table of Contents

No.	Content	Page No.
	Declaration	Error! Bookmark not defined.
	Acknowledgements	ii
	Abstract	iii
	List of Tables	viii
	List of Figures	ix
	List of Appendices	x
	Abbreviations	xi
	Chapter 1: Introduction	1
	1.1 Introduction	2
	1.2 Problem Statement.....	3
	1.3 Research Questions	3
	1.4 Research Motivation.....	4
	1.4.1. The Need for Analytical Intrusion Detection System	4
	1.4.2. Challenges of Analytical IDS	4
	1.4.3. Limitations of Existing Work	5
	1.5 Research Objectives	5
	1.6 Research Methodology	6
	1.7 Organization of the Thesis.....	7
	Chapter 2: Background and Related Work	9
	2.1 Background.....	9
	2.1.1. Information Security.....	9
	2.1.1.1. Basic Security Concepts	10

2.1.1.2. Basic Information Security Attack Vectors.....	11
2.1.1.3. Security Tools.....	12
2.1.1.3.1. Offensive Tools	12
2.1.1.3.2. Defensive Tools.....	14
2.1.1.4. National Vulnerability Database	15
2.1.2. Semantic Web.....	17
2.1.2.1. Ontology	18
2.1.2.1.1. Ontology Components.....	18
2.1.2.2. SPARQL.....	21
2.1.2.3. Inference Engine [Reasoner]	21
2.1.2.4. Jena Framework.....	21
2.1.2.5. EasyRdf Library	21
2.1.3. Google Visualization	22
2.1.4. Queuing Model.....	22
2.2 Related Work.....	24
2.2.1. Related Works for IDS	24
2.2.2. Related Works for Applying Semantic Web to Information Security	25
2.2.3. Related Work for Performance.....	27
2.2.4. Related work for information security.....	27
2.2.5. Summary.....	29
Chapter 3: Architecture.....	35
3.1 Introduction	35
3.2 System Components	36
3.2.1. Database Preparer	38
3.2.2. System Updater	38
3.2.3. Structure Updater.....	38
3.2.4. Categorizer.....	39
3.2.5. Determiner	39

3.2.6. Visualizer.....	40
3.3 Workflow process.....	40
3.3.1. System Initialization.....	40
3.4 Algorithm	43
3.4.1. Prepare the Inference Engine.....	45
3.4.1.1. Building the Ontology Skeleton.	45
3.4.1.2. Update the Database Signature of NVD.....	52
3.4.1.2.1. Detailed Algorithm.....	55
3.4.1.3. Update the System Structure	56
3.4.1.4. Input Data from Network Traffic and Extract CVE's	60
3.4.1.5. Reading CVE's and preparing the Data	62
3.4.1.5.1. Methods that Concerned About the Services	63
3.4.1.5.2. Methods that Concerned About the Related Attacks	65
3.4.1.5.3. Methods that Connected Directly to the Visualizer	66
3.4.1.6. Querying and Inference Necessary Information from OKB	66
3.4.1.6.1. Classes Related to Services	67
3.4.1.6.2. Classes Related to Attacks.....	68
3.4.1.7. Prepare Data to be Displayed and Visualized	68
3.4.1.7.1. Pie Chart Method.....	69
3.4.1.7.2. Annotation Chart Method.....	69
3.4.1.7.3. Word Trees Chart Method.....	69
3.4.1.7.4. Column Chart Method.....	69
3.4.1.7.5. Organization Chart Method.....	70
3.5 Summary.....	70

Chapter 4: System Validation 71

4.1 KDDCup99 Experimental Data.....	71
4.1.1. Measuring the Accuracy of attack prediction.....	72
4.1.1.1. Predictions based on two parameters	72

4.1.1.2. Predictions based on three parameters	73
4.1.1.3. Predictions based on four parameters	73
4.1.1.4. Comparing the results with other systems	74
4.2 Queuing Model	75
4.3 Simulating the System using Anylogic	79
4.4 System User Interface.....	82
4.4.1. Annotation Bar Page.....	82
4.4.2. Service Display Page	84
4.4.3. Dashboard Page	86
4.4.4. System Updates / Upgrade page.....	88
4.5 Summary.....	90
Chapter 5: Summary and Future Work	91
5.1 Contribution.....	91
5.1.1. Extracting Useful Information from Snort NIDS and NVD.....	92
5.1.2. Automatic Ontology Updates	92
5.2 Results	93
5.2.1. Measures using KDD Benchmark	93
5.2.2. Queuing Model	93
5.3 Limitations and Assumptions	94
5.4 Future Work.....	94
References.....	96
ملخص	104

List of Tables

Table No.	Table Header	Page No.
2.1-A	Summary of papers (1-2).....	29
2.1-B	Summary of papers (3-5)	30
2.1-C	Summary of papers (6-7).....	31
2.1-D	Summary of papers (8-10).....	32
2.1-E	Summary of papers (11-13)	33
4.1	Two parameters metric for accuracy prediction	72
4.2	Three parameters metric for accuracy prediction	73
4.3	Four parameters metric for accuracy prediction.	74
4.4	System comparisons.	74
4.5	Time Required to process different sizes of requests at same time.....	76

List of Figures

Figure No.	Figure Header	Page No.
2.1	IC3 cyber-criminal report	10
2.2	Sample NVD data feed	17
2.3	Sample RDF triple in XML/RDF format.	20
2.4	Sample of Google charts[Google].	22
2.5	Jackson closed network for multiprocessors server with caches [marl].	23
3.1	Placement of smart analysis engine.	36
3.2	System architecture.	37
3.3	Information flow chart in the proposed system.	42
3.4	Ontology skeleton.	46
3.5	Adding XML entries into built ontology.	54
3.6	Mechanism in structure update.	58
3.7	Input Data from Network Traffic and Extract CVE's.	62
4.1	System queue model.	77
4.2	Simulating the system using Anylogic.	81
4.3	Annotation bar page.	83
4.4	Service display page.	85
4.5	Dashboard page.	87
4.6	System updates / upgrades process.	89

List of Appendices

Appendix No.	Appendex Header	Page No.
1	Snort.....	101
2	Protégé.....	102

Abbreviations

AFRL	Air Force Research Laboratory
AI	Artificial Intelligent
BJA	Bureau of Justices Assistance
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DARPA	Defense Advanced Research Projects Agency
DMZ	Demilitarized Zone
FBI	Federal Bureau of Investigation
GA	Genetic Algorithm
GUI	Graphical User Interface
HIDS	Host Intrusion Detection System
IC3	Internet Crime Complaint Center
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
ISP	Internet Service Provider
IT	Information Technology
NIDS	Network Intrusion Detection System
NVD	National Vulnerability Database
NW3C	National White Collar Crime Center
OKB	Ontology Knowledge Base
OWL	Web Ontology Language

PHP	P ersonal H ome P age
RDF	R esource D escription F ramework
SCAP	S ecurity C ontent A utomation P rotocol
URI	U niform R esource I dentifier
XML	E xtensible M arkup L anguage

Chapter 1

Introduction

This chapter introduces the thesis. It describes the problem statement, research questions, motivations, goals and objectives, background, and organization of the thesis.

1.1 Introduction

With increasing needs for online services in the world, and the sensitivity of information transferred between peoples and organizations using the internet, the need for information security has become one of the most important issues for every person and organization. Many varieties of software and hardware's are used for the purpose of protecting the transferred information, such as firewalls, logging systems, Intrusion Prevention Systems [IPS], Intrusion Detection Systems [IDS], anti-virus programs, and others (Panda & Patra, 2007). Each one of these tools differs in its functionality and methods of protecting the network. The research we undertook concerns IDSs.

IDSs technologies are used for monitoring and logging attacks which occur in a single host or network. Due to the complexity of attacks occurred, such as SSH brute force attacks, web applications attacks, and other systems services attacks. Current IDSs suffers from the following issues:

- Signature based IDS, cannot predict new attacks.
- Signature based IDS, cannot manage and connect huge amount of data logged.
- Anomaly based IDS, requires a data to be trained to predict attacks.

- Anomaly based IDS, produces large false positive rate in detection.

For this, the need for analytical IDS is required. Which should combine the advantages of both signature and anomaly based IDSs, and minimize the disadvantages of both.

1.2 Problem Statement

Currently, security systems such as intrusion detection systems manage engines [for log management] do not provide rich analytics on security threats and vulnerabilities for system administrators (Kotikela, Kavi , Gomathisankaran, & Singhal, 2013) (D. Kshirsagar & Kumar, 2013). For enterprise environments, such as ISPs, schools, and universities the Demilitarized Zone [DMZ] contains several critical servers that are always targeted. The current tools, for instance, do not predict attacks that may be launched against DMZ and cause the servers to be compromised. Therefore, there is a high need for more security analytics and monitoring software. In other words, the security analytics system should do the following:

- Predict related attacks that may hit the IT systems.
- Predict systems in the DMZ that may be compromised.
- Show the level of risk each attack may cause.

Numbers of attacks that may be launched to compromise a system are huge, due to the increasing number of automated hacking tools: such as METASPLOIT, CANVAS, CORE IMPACT, and others (nmap CO., 2016). These hacking tools depend on the services opened to lunch their remote attacks. In addition, each tool contains a huge database that automates different types of attacks against single system service (Bairwa, Mewara, & Gajrani, 2014). A normal IDS [signature based or anomaly based] that exists in the network cant neither predict what attacks may be automated to a single service, nor draw a relation between existing

systems in the network, to perform a prediction on other systems or services that may be in danger. In other words, these IDSs only log the attacks that occurred [signature based], or from the behavior of the network traffic, to predict if a packet is an attack or not [anomaly based]. So there are no smart IDS that are able to log security information from network traffic, and do a prediction on the logged traffic. From there the idea of semantic analysis features for IDS is created.

1.3 Research Questions

The following questions should be answered by our proposed system:

- What are the attacks that may hit the network systems and make them vulnerable?
- What are the classifications of attacks that targeted network systems?
- What are the network systems that may be targeted in the future?
- How the system will display statistics and security information to the administrator?
- How the system will adapt to continuous changes in the network infrastructure?

1.4 Research Motivation

1.4.1. The Need for Analytical Intrusion Detection System

Analyzing log files is very important to detect and correct errors in systems, especially security errors. Many software systems, firewalls and Intrusion Prevention Systems [IPS] or Intrusion Detection Systems [IDS], show only information about current attacks, attacks on a specific date or signature or predict behavior of an attack. According to our knowledge, there is no software that does semantic analytics and prediction on security threats and vulnerabilities. So resolving these obstacles in smart IDS should produce the following benefits:

- Adding analytical features to the IDS.
- Reducing the possibility of false alarms, since the predictions are based on attributes.
- Providing a better understanding of the data stored.
- Providing a dynamic system that can be adapted according to the needs of the user environment, by allowing the user to add new system service dynamically. So the attacks prediction becomes more accurate.

1.4.2. Challenges of Analytical IDS

Our proposed IDS consists of two parts: the logger system, and the analyzer system. These parts should be able to communicate with each other's. The analytical part of the system consists of multiple parts. Each part has its own functionality, and complements the others.

Due to the nature of data that will be entered and presented, different programming languages will be used, so that the connection between these components should be smooth. Furthermore, the continuous changes in the needs of network services should be reflected easily in the system, and the display of results must be easily understandable.

1.4.3. Limitations of Existing Work

Two types of IDSs exist: anomaly based IDS, and signature based IDS. Signature based IDS, only reads the payload of a packet, and then checks it in its database, to determine if this payload contains a signature of an attack or not. Anomaly based IDS examines the behaviors of packets in network traffic, based on trained data before, to determine if the examined packets payloads are normal, or containing security threats. This may produce a lot of false alarms. For these reasons, smart IDS should resolve the issues of these IDSs. The new IDS should take advantages of both types, by using a signature to log attacks, and perform predictions on new attacks from the logged attacks without the need for data training.

1.5 Research Objectives

The overall goal of this thesis is to improve the IDS, and develop a new objective function to propose a new analytical approach for IDS. We refer to our new approach as ‘Smart Intrusion Detection System for DMZ’, and it aims to introduce a new IDS with analytical features, good minimized false rate attacks predictions, and a more easily adaptable IDS according to the network needs. To achieve this overall goal, the following research objectives have been established:

- Extract useful and needed security information from network traffic.

- Create ontology for vulnerabilities identification.
- Automate the creation of ontology structure [schema], to reflect the changes according to the needs of the user.
- Automate the creation of Ontology Knowledge Base [OKB], from vulnerabilities databases.
- Map the extracted security information from network traffic to the created OKB, so that additional security information can be extracted.
- Use a visualization technique, to display the results in an easily readable view.
- Validate the system using known benchmarks.
- Predict the behaviors of the system in a multi-user heavy network environment, using a queuing model.

1.6 Research Methodology

This section describes the research methodology that was followed.

- Plan constructing our system using modular forms. This ensures that it can be easily updated, and the changes can be smoothly performed.
- Create an ontology skeleton that reflects vulnerabilities and network services, taking into account that the ontology should be changed, according to the demand of the network services.

- Develop OKB that we will use in analysis by extracting National Vulnerability Databases [NVD] into the created ontology. We should take into accounts that NVD databases continuously change.
- Conduct a method to extract security information from network traffic. After that, convert the extracted security information, into information that can be understandable by the created OKB.
- Develop communication protocols between the sniffed security information from snort IDS and the created OKB.
- Analyze the information sent from the network, by performing extra processing on it in the, by the created OKB. The analysis should be performed by semantic web tools, so that it will reason and query information related to information extracted from network traffic.
- Develop a way that presents the results in a nice Graphical User Interface [GUI]. So the results with huge rich information should be easily understandable.
- Conduct experiments, by applying the proposed system in a known benchmark, to ensure the effectiveness of it.
- Model the system in queuing model, to predict the behaviors of the system in a multi-user heavy network environment.

1.7 Organization of the Thesis

This chapter gives an overview of the thesis. It presents the basic concepts of information security, IDSs, performance and semantic web technology that should be understandable. The

next chapter is literature review, which reviews traditional related works in information security, IDSs, performance, datamining, and focuses on ways to create smart security tools using semantic web technology. Chapter 3 introduces system architecture. This chapter proposes a new novel approach, which improves the analytical feature in IDS, and develops new functionality to the IDS, based on semantic technology. The validation and results will be discussed in chapter 4. It examines our approach, and conducting three approaches to measure it. The first approach is by testing the system on KDDCup99 dataset and comparing its results with other systems. A queuing model is the second approach. Anylogic is the final way, to simulate and validate the queuing model. Finally, the results are discussed. Chapter 5 is the conclusion, which discusses the conclusions of the thesis, limitations and assumptions, and also suggests some possible future work.

Chapter 2

Background and Related Work

2.1 Background

This section aims to provide a general discussion of the concepts needed to understand the rest of the thesis. It covers basic concepts of information security, IDSs, ontology, Google visualization, and queuing models.

2.1.1. Information Security

In our days, organizations depend greatly on networks, a huge amount of information is exchanged through networks. Daily tasks performed rely on computers and networks, such as email, portal services, RSS, and others. However, losing or revealing this information may cause a terrible loss for an organization. Consequently, there is a sense of urgency to secure electronic information.

Information security refers to protecting any kind of sensitive information systems from being revealed by unauthorized access (Cornell University) (Wikipedia, 2016). For most people and organizations, electronic information is a critical resource to be protected. On the other hand, if sensitive information is revealed to the public or the wrong person, then an organization may face a great threat, and the whole business may be in danger. For instance, if sensitive database system for an organization is hacked, and such information falls in wrong hands, it can create chaos in the normal functioning of an organization.

Figure (2.1) is the cyber-criminal report data from IC3; the Internet Crime Complaint Center [IC3] is a partnership among the federal Bureau of Investigation [FBI], the National White Collar Crime Center [NW3C], and Bureau of Justice Assistance [BJA]. According to IC3,

online internet crime complaints are increasing daily. From the figure, we can observe that in year 2010, there were 303,809 cyber-crime complaints, whereas in year 2011, complaints increased to 314,246. When compared to 2011, Internet crime complaints in year 2013 decreased to some extent. But in subsequent year the internet crime compliant increased (IC3, 2014).

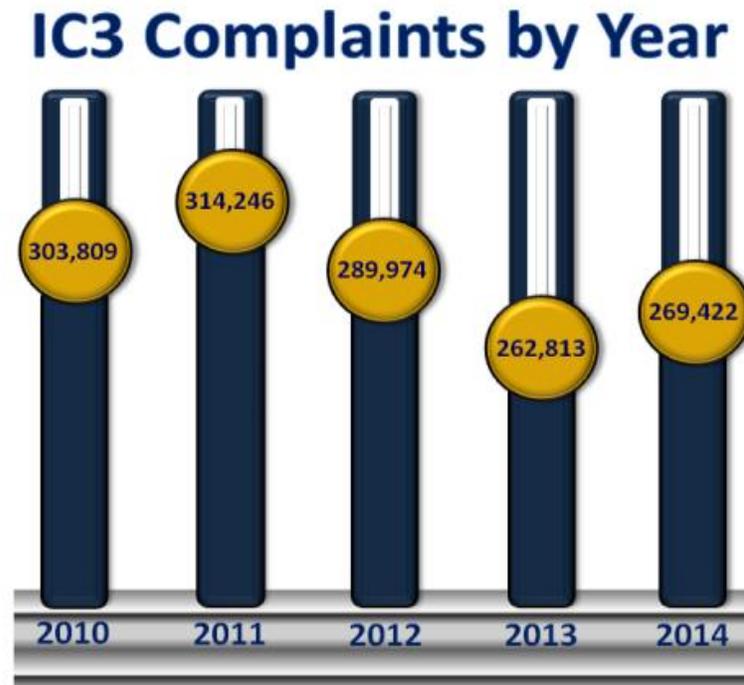


Figure 2.1: IC3 cyber-criminal report (IC3, 2014).

2.1.1.1. Basic Security Concepts

This section summarizes basic security concepts necessary to understand our research.

- **Vulnerability:** a weakness in design or an implementation error that can lead to compromising the security of the system. In other words, vulnerability is a loop hole, or a weakness that becomes a source for an attacker to enter into the system, bypassing

various user authentications (Wang & Guo, OVM: An Ontology for Vulnerability Management, 2009).

- **Exploit:** a defined way to break the security of a system through vulnerability. An exploit can be performed locally or remotely through a network.

- Information security relies on three basic elements (Parker, 1995), which are:
 - **Confidentiality and Authenticity:** to ensure that the information is accessible to a person who is authorized to access these data, by using certain authentication methods (Parker, 1995).

 - **Integrity:** the trust of data or resources in terms of preventing unauthorized changes (Parker, 1995).

 - **Availability:** to ensure that a system is available and online whenever a service request is performed.

2.1.1.2. Basic Information Security Attack Vectors

The following are possible attack vectors which attackers can exploit an information system:

- **Unpatched software:** where the system or software left not updated for a long time, for which a lot of vulnerabilities exploits had been published against it (Ec-Council, 2010).

- **Local attacker:** where the attacker has a good knowledge of the systems exists in an organization. For example, a descrambled employee that has access to the financial system with authorized username and password (C. C. Palmer; IBM Research Divisio, 2001).

- **Network applications or services:** where the attacks lunched from outside of an organization targeting a system online services. Such as, company website (C. C. Palmer; IBM Research Divisio, 2001).
- **Botnets:** where group of attackers lunched several attacks from different locations targeting a system service, to hack the system or to bring it down (Ec-Council, 2010).
- **Insufficient security policies:** where the systems administration privilege is distributed among different users (Ec-Council, 2010).
- **Social networking:** where there are not educated employees from security perspectives can be easily fooled by attackers. For instance, email sent to an employee to fill his username and password for organization portal in a form sent by that email (C. C. Palmer; IBM Research Divisio, 2001).

2.1.1.3. Security Tools

Two types of tools are used in information security, offensive and defensive tools.

2.1.1.3.1. Offensive Tools

Offensive tools are the kinds of tools used by hacker[s] or security tester[s] to discover, and exploits the vulnerabilities of information systems. Each professional attacker follows five steps to perform a successful exploitation into information system. These steps are:

- **Information gathering [reconnaissance]:** in which an attacker retrieves general information about the targeted organization (Phong & Yan, 2014).

- **Scanning and vulnerability analysis:** in this phase the attacker actively tests the systems available for an organization, to determine the type of services that are available online and type of vulnerabilities that might exploit these services (Phong & Yan, 2014) (Bairwa, Mewara, & Gajrani, 2014).
- **Exploiting:** the attacker will run various exploits on systems services, based on previous steps performed (Phong & Yan, 2014).
- **Root access:** after successfully exploiting the required IT system, the attacker tries to get administrator privilege on that hacked system, in order to attain full control over it (Ec-Council, 2010).
- **Log erase:** the final steps performed by a professional attacker are to hide his or her footprint on the hacked systems by installing malicious software's on it, such as a rootkit (Ec-Council, 2010).

Offensive Hacking tools can be classified into three main categories:

- **Information gathering tools:** tools that are used to get information about IT systems, such as the location, registrar, and subdomains. An example of these tools is WHOIS (Ec-Council, 2010) (Phong & Yan, 2014).
- **Scanning tools:** tools are used to get information about online IT systems. They return information about the system type, services opened, version of opened services, and other useful information. NMAP is a good example of scanning tools (Bairwa, Mewara, & Gajrani, 2014) (Phong & Yan, 2014).
- **Exploiting tools:** tools are used to penetrate the information system and gain unauthorized access to it. They automate the attacks into a system based on services provided. Examples of these tools are CANVAS, METASPLOIT and CORE-IMPACT (Ec-Council, 2010) (Phong & Yan, 2014).

2.1.1.3.2. Defensive Tools

The information system, especially the enterprise network and DMZ, containing the heart of electronic information, should be protected. As the security increased the robust our system is. On the other hand, we should take into account the usability of the system. Several varieties of systems hardware or software can used to protect information systems in their own way. The following subsections summarize some of the important security tools that an organization can use.

Firewall: This tool should be the first layer of defense in any network. It may be hardware or software. Furthermore, it might be commercial or open source. Today, these tools are smart they inspect packet in all network layers, in other word they are state full inspection. Examples of hardware commercial firewalls are: FOTINET and SONICALL. IPABLE is an example of a software open source firewall.

Intrusion Prevention System [IPS]: Security tool that inspects network traffic, to detect malicious behavior in it, then block it and report it (Wikipedia, 2016).

Antivirus: Software that is installed on a machine, to detect and protect it from malware, that attempts to steal or destroy the information on that system.

Intrusion Detection System [IDS]: Security tool that might be software or hardware, which is used to inspect network traffic. An IDS examines the packets inside network traffic, to check if it contains any malware packets, then reports the results (Panda & Patra, 2007). Two types of IDSs are available:

- **Host based IDS (HIDS):** software that evaluates information which exists on single host or multiple hosts (D. Kshirsagar & Kumar, 2013).
- **Network IDS (NIDS):** software or hardware. NIDS evaluates and analyzes information captured from network traffic. On other words, it monitors the network

activity for a malicious behavior, logs it, and reports it to the administrator (D. Kshirsagar & Kumar, 2013).

Furthermore, IDSs can be classified into:

- **Signature based IDS:** An IDS that is used to detect known attacks. It stores the attacks signature on an IDS database. This type of IDS requires no knowledge of network traffic: the IDS will only check the payloads in packets, then compare it in its database, to determine if these packets are attacks or not. If they are malicious, then the result will be logged and reported (Uddin, Khowaja, & Abdul Rehman, 2010).
- **Anomaly based IDS:** An IDS where the signature of an attack is not known before. These types of IDSs are usually used to detect new types of attacks, based on previous data behavior (Uddin, Khowaja, & Abdul Rehman, 2010). In other words, it requires data to be trained, so that it can detect new attacks. A lot of research is performed on these types of IDSs, specifically to discover zero day attacks. In general these IDSs are created using smart algorithms, such as Genetic Algorithm [GA], and datamining methods, such as naive bays.

Snort is an example of a signature based NIDS. It is considered the top open source intrusion detection in the world.

2.1.1.4. National Vulnerability Database

“NVD is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol [SCAP]” (Quinn, Waltermire, Johnson, Scarfone, & Banghart, 2009). NVD contains rich data about vulnerabilities, and useful attack information. The following is a summary of the important content of NVD:

- **Vulnerability information:** contains normal information about the vulnerability, such as the date and name of the vulnerability.

- **Product:** contains a list of all products that are affected by a given vulnerability. The product name is in CPE format.
- **Vulnerability score:** contains a numeric number, stating how dangerous the vulnerability is.
- **Summary:** contains useful information about how an attacker can exploit a given vulnerability. In addition, it contains information of damage that can occur, if a successful attack occurs.

Figure (2.2) shows a sample of NVD data feeds.

```

<entry id="CVE-2015-0075">
  <vuln:vulnerable-configuration id="http://www.nist.gov/">
    <cpe-lang:logical-test operator="OR" negate="false">
      <cpe-lang:fact-ref name="cpe:/o:microsoft:windows_2003_server::sp2"/>
    </cpe-lang:logical-test>
  </vuln:vulnerable-configuration>
  <vuln:vulnerable-software-list>
    <vuln:product>cpe:/o:microsoft:windows_server_2008::sp2</vuln:product>
  </vuln:vulnerable-software-list>
  <vuln:cve-id>CVE-2015-0075</vuln:cve-id>
  <vuln:published-datetime>2015-03-11T06:59:04.980-04:00</vuln:published-datetime>
  <vuln:last-modified-datetime>2015-03-11T12:03:47.810-04:00</vuln:last-modified-datetime>
  <vuln:cvss>
    <cvss:base_metrics>
      <cvss:score>7.2</cvss:score>
      <cvss:access-vector>LOCAL</cvss:access-vector>
      <cvss:access-complexity>LOW</cvss:access-complexity>
      <cvss:authentication>NONE</cvss:authentication>
      <cvss:confidentiality-impact>COMPLETE</cvss:confidentiality-impact>
      <cvss:integrity-impact>COMPLETE</cvss:integrity-impact>
      <cvss:availability-impact>COMPLETE</cvss:availability-impact>
      <cvss:source>http://nvd.nist.gov</cvss:source>
      <cvss:generated-on-datetime>2015-03-11T11:41:06.020-04:00</cvss:generated-on-datetime>
    </cvss:base_metrics>
  </vuln:cvss>
  <vuln:cwe id="CWE-264"/>
  <vuln:references xml:lang="en" reference_type="VENDOR_ADVISORY">
    <vuln:source>MS</vuln:source>
    <vuln:reference href="http://technet.microsoft.com/security/bulletin/MS15-025" xml:lang="en">MS15-025</vuln:reference>
  </vuln:references>
  <vuln:summary>The kernel in Microsoft Windows Server 2003 SP2, Windows Vista SP2, Windows Server 2008 SP2 k</vuln:summary>
</entry>

```

Figure 2.2: Sample NVD data feed

2.1.2. Semantic Web

This section focuses on the semantic web technology, as well as the technologies that are used in our research; such as Jena Java framework and easyRDF PHP library.

2.1.2.1. Ontology

“Ontology is a formal explicit specification of a shared conceptualization” (Semantic Web, 2012). In a modern smart system, knowledge should be discovered and the ontology technology should perform this task (Wang & Guo, OVM: An Ontology for Vulnerability Management, 2009). Ontologies solve the problem of semantic interpretation between different web services in a different organization through semantic web technologies. Recently, ontologies have become a popular field of research in Artificial Intelligence [AI], and knowledge discovery systems, such as modern social networks (Taye , 2010). Semantic web is one of the applications that are built on the top ontology technology, to make the web machine understandable, not just machine readable.

2.1.2.1.1. Ontology Components

Ontology should be represented using the following four main components:

- **Concept (Class):** an abstract set of objects.
- **Instance (Individual):** the “ground-level component of an ontology which represents a specific object or element of a concept or class” (Taye , 2010).
- **Relation (Slot):** used to connect two classes for a given domain.
- **Axiom:** used to give a constraint on a value of a concept or individual.

Usually ontology is stored in a triple format file. This file may be in turtle format: which is the most readable format, or in XML/RDF, or Notation 3, or RDF/JSON, or N-Triples, or JSONS formats. Any triple inside a file [in any format], is made up of three parts: subject, predicate and object, which form a statement. A subject of a statement is an entity that the statement describes. A predicate describes a relationship between a subject and an object. The object is

the value that the subject takes for that particular predicate. These statements inside a file form a directed graph, with subjects and objects of each statement as nodes and predicates as edges (Semantic Web, 2012).

The RDF or OWL files are the most common extensions of semantic web files. Each file consists of two parts. Firstly, there is the schema [T-Box] part, which represents the concepts and relations. In other words, it represents the structure of ontology. The other part of ontology is the data [A-Box] part, which represents the instances. Figure (2.3) shows an example of RDF graph in XML/RDF format.

```

<rdf:Description rdf:about="http://scap.nist.gov/schema/vulnerability/vuln-CVE-1999-0012/">
  <rdf:type rdf:resource="http://www.owl-ontologies.com/Ontology1353426448.owl#Vulnerability"/>
  <rdfs:label>CVE-1999-0012</rdfs:label>
  <ns0:hasVulnerabilityName>CVE-1999-0012</ns0:hasVulnerabilityName>
  <ns0:hasVulnerabilityCVEID>CVE-1999-0012</ns0:hasVulnerabilityCVEID>
  <ns0:hasVulnerabilityPublishedDate>1998-02-06T00:00:00.000-05:00</ns0:hasVulnerabilityPublishedDate>
  <ns0:hasVulnerabilityDescription>Some web servers under Microsoft Windows allow remote attackers to bypass access
    restrictions for files with long file names.
  </ns0:hasVulnerabilityDescription>
  <ns0:hasAttack>
    <ns0:Attack rdf:about="http://scap.nist.gov/schema/vulnerability/vuln-CVE-1999-0012/Attack/">
      <rdfs:label>Attack_CVE-1999-0012</rdfs:label>
    </ns0:Attack>
  </ns0:hasAttack>

  <ns0:hasAttackMechanism>
    <ns0:AttackMechanism rdf:about="http://scap.nist.gov/schema/vulnerability/vuln-CVE-1999-0012/AttackMechanism/">
      <rdfs:label>AttackMechanism_CVE-1999-0012</rdfs:label>
      <ns0:hasActiveLocation> bypass access restrictions for files with long file names.</ns0:hasActiveLocation>
    </ns0:AttackMechanism>
  </ns0:hasAttackMechanism>

  <ns0:hasAttacker rdf:resource="http://scap.nist.gov/schema/vulnerability/vuln-CVE-1999-0012/Attacker/">
  <ns0:hasCVSSMetrics>
    <ns0:CVSSMed rdf:about="http://scap.nist.gov/schema/vulnerability/vuln-CVE-1999-0012/CVSSMetricCVSSMed/">
      <rdfs:label>CVSSMetric_CVE-1999-0012</rdfs:label>
      <ns0:hasCvssAccessComplexity>LOW</ns0:hasCvssAccessComplexity>
      <ns0:hasCvssAccessVector>NETWORK</ns0:hasCvssAccessVector>
      <ns0:hasCvssAuthentication>NONE</ns0:hasCvssAuthentication>
      <ns0:hasCvssAvailabilityImpact>NONE</ns0:hasCvssAvailabilityImpact>
      <ns0:hasCvssConfidentiality>PARTIAL</ns0:hasCvssConfidentiality>
      <ns0:hasCvssGeneratedOnDateTime>2004-01-01T00:00:00.000-05:00</ns0:hasCvssGeneratedOnDateTime>
      <ns0:hasCvssIntegrityImpact>NONE</ns0:hasCvssIntegrityImpact>
      <ns0:hasCvssScore>5.0</ns0:hasCvssScore>
    </ns0:CVSSMed>
  </ns0:hasCVSSMetrics>

  <ns0:isAffectedITProducts>
    <ns0:frontpage rdf:about="http://scap.nist.gov/schema/vulnerability/0-vuln-CVE-1999-0012/AffectedITProductsApp/">
      <rdfs:label>ITProductApp_CVE-1999-0012</rdfs:label>
      <ns0:isITProduct>cpe:/a:microsoft:frontpage</ns0:isITProduct>
    </ns0:frontpage>
  </ns0:isAffectedITProducts>
</rdf:Description>

```

Figure 2.3: Sample RDF triple in XML/RDF format.

2.1.2.2. SPARQL

A tool that is used to query data stored in RDF or OWL file, to get required information about the stored triple.

2.1.2.3. Inference Engine [Reasoner]

An inference engine is used to extract additional factual from instance information. It automates the analysis of data contents. Also, it may be used to ensure our data graph is consistent (W3C org, 2015). Examples of reasoners are Hermit and Pellet.

2.1.2.4. Jena Framework

An open source java framework dedicated to dealing with semantic web technologies. It has APIs that are able to create, change, and query RDF triples stored in any format. In addition, Jena has a strong built-in inference engine, and further-more supports known reasoners, such as Pellet (Apache Jena, 2015).

2.1.2.5. EasyRdf Library

A PHP open source library, which can read and add RDF triples over HTTP. It has vast APIs that allow the reading of stored RDFs easier over the web. On the contrary, it has limitations on performing SPARQL queries, and using of reasoners.

2.1.3. Google Visualization

A free cross platform tool that is used to visualize data over the web. It can be used by including the visualization APIs inside a JavaScript script. It has many charts types; such as: pie, annotation bar, histogram, organizational charts and many more. These charts are interactive, and can be customized easily. Figure (2.4) shows some examples of charts of this rich tool (Google org, 2015).



Figure 2.4: Sample of Google charts (Google org, 2015).

2.1.4. Queuing Model

"Is the mathematical study of waiting lines or queues". We use queuing models to predict waiting times, and queue length before getting a service (Wikipedia, 2016). In each queue model design in any field, the following information should be known: arrival time, queue size, queue type, number of servers and service time. Queuing model can be categorized into:

single queue and multiple queues. Single queue is used if we have only one place for getting a service, such as modeling a doctor clinic. We use multiple queues if there are multiple services in a place, for example if we want to model multiprocessors with caches server. A Jackson network is an example of a multiple queue models; it is the best way to model this scenario. Figure (2.5) shows an example of Jackson closed network (Belch, Greiner, de Meer, & Trivedi, Chapter 13. Applications, 1998).

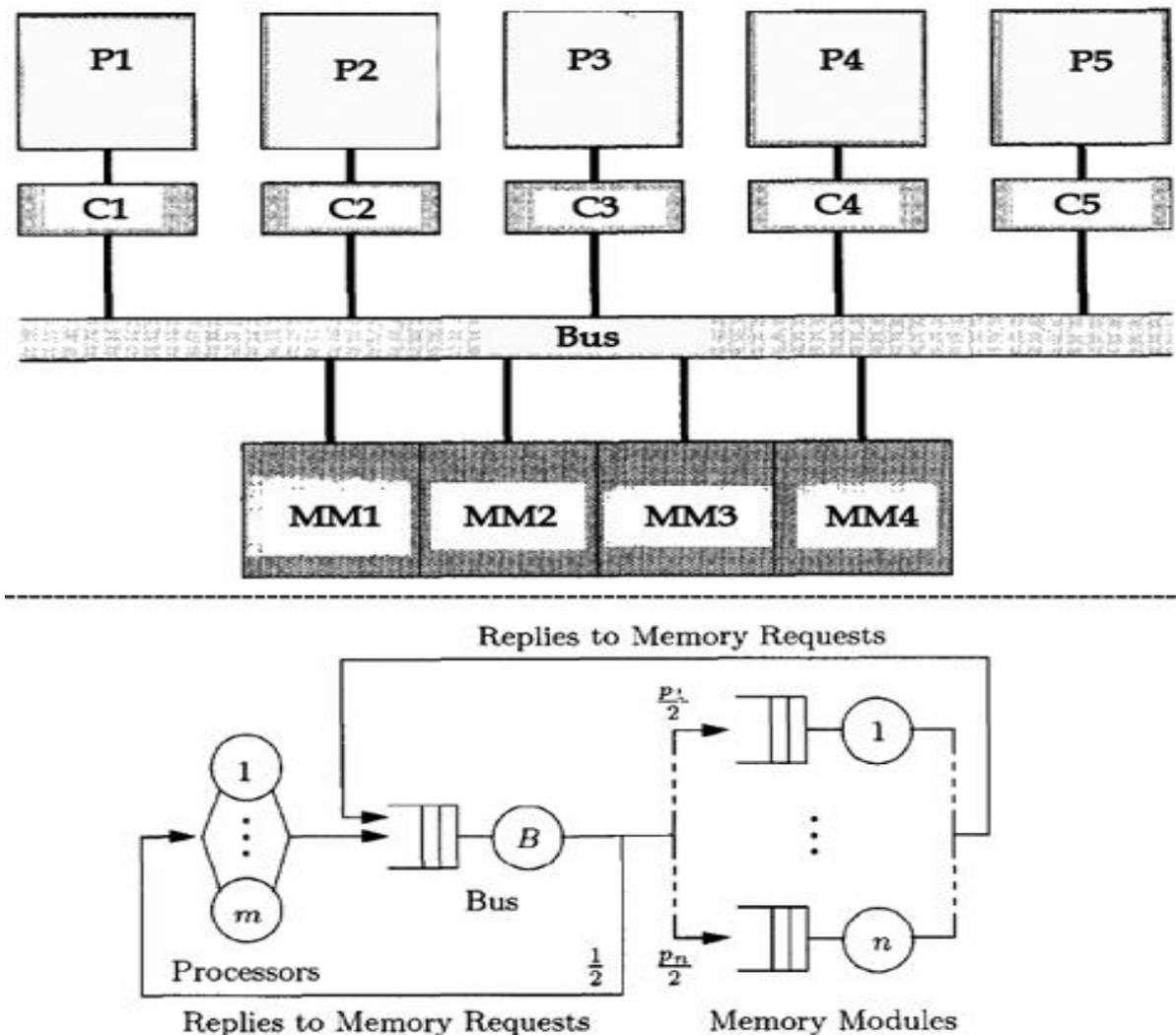


Figure 2.5: Jackson closed network for multiprocessors server with caches (Belch, Greiner, de Meer, & Trivedi, Chapter 13. Applications, 1998).

2.2 Related Work

Several researches have been published in the area of information security, ontology, datamining, semantic technology, Intrusion Detection Systems [IDSs] and applying these technologies to information security. This chapter reviews related works for IDSs, semantic web technology for security, security software's algorithms, and performance measuring. The following subsections summarize these related works.

2.2.1. Related Works for IDS

There are many varieties of IDS software and they are widely used. Some of them are open source and others are commercial. In addition there are two common types of IDSs: that might use genetic algorithm, or data mining, or database signature to discover attacks. In our research we used snort IDS to detect and log attacks from network traffic .The following papers summarized these types of IDSs.

Panda et al (Panda & Patra, 2007), proposed anomaly based intrusion detection system [IDS]. The authors showed how to implement Naïve Biase classifier, to detect new attacks. However, the proposed IDS is restricted to a network that has only two levels and assumes complete independency between the information nodes. Also for this research, data was required to be trained, to detect new attacks. This study produced an anomaly based IDS that has a goal to enhance detection rate of newly unknown attacks. In our research, we used signature-based detection to detect and log attacks, and for the analysis engine no data training was required, to perform attacks predictions.

Hoque et al (Hoque, Mukit, & Bikas, 2012), created an intrusion detection system that is based on Genetic Algorithms (GA). The implementation of this IDS was performed in two steps. The first step is: the precalculation phase, where data will be trained. The second step is detection phase. Controversially, using GA showed a lot of false positive alarm in the results.

In our research the rate of false alarms and accuracy of attacks predictions were minimal. In addition, there was no need for the data training.

2.2.2. Related Works for Applying Semantic Web to Information Security

Semantic web in our research represent the heart of it, from it we can predict and extract additional useful information about vulnerabilities attacks. We created Ontology Knowledge Base [OKB] from National Vulnerability Database [NVD], which we used in our analysis system. Many of the following studies showed us how useful it is to use ontology in the field of information security. Some of this research represents a way to create ontology for NVD. But none of them are specialized for DMZ networks. Also, none of the following research contains automatic ontology updates. The following research represents the importance and usefulness of semantic web in information security:

In Gomathisankaran et al (Kotikela, Kavi , Gomathisankaran, & Singhal, 2013), the authors worked on providing a smart vulnerability assessment framework for cloud computing. In this study, NVD was converted to knowledge base that are dedicated for cloud computing; also a web search was made on top of these vulnerabilities to get more information about the vulnerability; Such as countermeasure for each vulnerability. However, the limitation of this framework that it was neither validated nor did compared its results with other known vulnerability assessment tools. The research presented a vulnerability assessment framework that uses ontology for cloud computing. In our research, we also used ontology to group and classify vulnerabilities for the systems services in the DMZ.

Khairkar A.D (D. Kshirsagar & Kumar, 2013) provided an intrusion detection system for detecting web attacks. In this IDS, ontology was used to group and classify web attacks. Furthermore, they used the ontology to identify new web attacks. On the other hand, the limitation of the proposed system was that it's unable to detect complex web attacks, especially for complex e-business systems. In this research ontology was used to classify web

attacks only. However, in our research ontology was used to classify and group attacks for different systems and services including known web attacks.

Wang et al (Wang & Guo, OVM: An Ontology for Vulnerability Management, 2009) , focused their research on providing semantic meaning to vulnerability database, and applying network security standard; such as CWE, CPE, CAPE, and CVE to build vulnerability ontology. On the other hand, generating data from vulnerability database was done manually using protégé tool, instead of extracting the data automatically. This is considered as a limitation. In our research, classes and relations for describing vulnerabilities were produced automatically, and the structure of the ontology can be changed automatically.

Wang et al (Wang, Guo, Wang, Xia, & Zhou, 2009) proposed an ontology based approach, to analyze and assets the situation of products from security perspective. The authors implemented a mathematical formula to compute each vulnerability score, based on how it is danger and its affection on the system or service.

In Saad et al (Saad & Traore, 2010), the authors provided a prototype for smart network forensic tools. In this paper, two types of relations were used in building the network forensic ontology: the taxonomic relation, and the ontological relation. In addition, three types of knowledge specified to be presented: problem solving goals, problem solving knowledge, and Factual knowledge. However, this prototype had one limitation in term of ontology creation; where there was huge ontology construction was performed manually, to reflect every possible scenario. On the contrary, in our research, ontology updates were performed automatically.

Salini et al (Salini & Shenbagam, 2015) constructed ontology to predict and classify web attacks. Also their ontology system suggests countermeasure for the predicted attacks. On the other hand, their ontology is created manually, and the system used only to detect web attacks. Conversely, in our system ontology creation is automated, and our system includes most common network services, including common web attacks.

In Elahi et al (Elahi, Yu, & Zannone, 2009), they proposed ontology to identify vulnerabilities, also they showed the impact of these vulnerabilities on systems and services on their ontology model. However, the ontology creation was made manually, and the results of attack prediction contain a lot of false alarm. On the contrary, ontology on our system is updated automatically, and the rate of false alarm attacks prediction is minimal.

2.2.3. Related Work for Performance

The following papers show different ways to improve the performance of IDSs. Some of these techniques may be partially implemented in our system, and other advanced techniques can be considered in the future work.

Uddin et al (Uddin, Khowaja, & Abdul Rehman, 2010), improved the performance of IDS by splitting the IDS into multiple IDSs. Each IDS is responsible for detecting and analyzing only specific attacks signatures. In this study multiple IDSs devices were required for detecting and analyzing attacks. This research has a limitation, in that it required a separate device for each class of attacks. As the number of attacks classification increases, this research will be impractical in the future.

Bulajoul et al (Bulajoul , James, & Pannu, 2013) , the main goal of this research was to show that at some points NIDS will not be able to analyze and log network traffic, especially for heavy network traffic with huge number of packets send through it. At the end of this research the authors recommend using parallel IDSs, to enhance the number of packets captured and analyzed.

2.2.4. Related work for information security

Bairwa et al (Bairwa, Mewara, & Gajrani, 2014), the authors showed the power of known vulnerability scanning tools, such as NEKTO and NESSUS tools. NEKTO tool is considered

one of the best web vulnerability scanning tools, whereas, NESSUS tool, is one of the best known network vulnerability scanning tools (nmap CO., 2016). In their research they showed how to use these tools in an effective way, to get the desired results.

2.2.5. Summary

Table (2.1-A), table (2.1-B), table (2.1-C), table (2.1-D), and table (2.1-E) summarize these papers and show a short description of each one of these researches.

Table 2.1-A: Summary of papers (1-2).

#	Reference id	Setting	Description of study design methodology	Type of research	Limitations	Intervention	Results	Comments
1.	Panda et al,2007	IDS	<ul style="list-style-type: none"> Apply naïve bias classifier to detect anomaly based intrusion. 	Empirical	Restricted to a network that has only two levels and assumes complete independency between the information nodes. Requires data to be trained.	Naïve classifier is used in attack detection.	Anomaly based intrusion detection system.	
2.	Hoque et al, 2012	IDS	<ul style="list-style-type: none"> Implements GA to IDS. In the implementation two phases performed by the IDS: the precalculation phase and detection phase. 	Empirical	Require data to train in earlier stage.	How to use GA to detect novel attacks.	Anomaly based intrusion detection system.	In our work we don't need to train data set to make a prediction.

Table 2.1-B: Summary of papers (3-5).

#	Reference id	Setting	Description of study design methodology	Type of research	Limitations	Intervention	Results	Comments
3.	Gomathisan et al, 2013	Semantic web	<ul style="list-style-type: none"> Convert national vulnerability database into OKB. Do a web search on each vulnerability. 	Empirical	This framework is neither validated nor compares its results with other known vulnerability assessment tools.	Using knowledge base in vulnerability assessment tool for cloud computing.	Vulnerability assessment tool for cloud computing that uses ontology.	
4.	Khairkar A.D,2013	Semantic web	<ul style="list-style-type: none"> Build ontology to classify web attacks. Use the created ontology to identify zero day web attack. Reduce false positive rate and high detection rate. 	Empirical	Unable to detect complex web attack especially for complex e-business system.	Using ontology knowledge base to detect web attacks.	Intrusion detection system for web.	
5.	Wang et al,2009	Semantic web	<ul style="list-style-type: none"> Study standard network security classifier such as CWE, CPE, and CAPE. Apply these standards to build ontology knowledge base. 	Empirical	No automatic way to extract data from vulnerability database.	Using standard security classifier to build ontology knowledge base.	Ontology knowledge base that classify vulnerabilities.	

Table 2.1-C: Summary of papers (6-7).

#	Reference id	Setting	Description of study design methodology	Type of research	Limitations	Intervention	Results	Comments
6.	Wang et al,2009	Semantic web	<ul style="list-style-type: none"> ontology approach to compute security metric 	Analytical		Formula used to compute score for each vulnerability based on how it is danger	Ontological approach for security metric.	
7.	Saad et al, 2010	Semantic web	<ul style="list-style-type: none"> Building forensic security tool based on ontology based on taxonomic relation and ontological relation. Three types of knowledge specified to be presented: problem solving goals, problem solving knowledge and factual knowledge. 	Empirical	Ontology construction is done manually.	Using ontology technology to build security forensic tool.	Prototype of smart forensic tool.	Good for future work.

Table 2.1-D: Summary of papers (8-10).

#	Reference id	Setting	Description of study design methodology	Type of research	Limitations	Intervention	Results	Comments
8.	Wu et al, 2009	Performance	<ul style="list-style-type: none"> Intrusion detection system that uses two devices: one for monitoring the traffic, capturing packets and decoding packets, and the other device for attack detection and analysis. 	Empirical	This research shows how to improve the performance but it does not add anything new in detection or analysis.	Distribute the services of intrusion detection system.	Intrusion detection system.	
9.	Uddin et al, 2010	Performance	<ul style="list-style-type: none"> Splitting the IDS into multiple IDSs, so that each IDS is responsible for detecting and analysis only specific attacks signatures. 	Empirical	Multiple IDSs devices are required. As the number of attacks signatures increases over time, we think this is not a practical design of IDS.	Enhancing the speed of detection and analysis of intrusion detection system.	Intrusion detection system.	
10.	Bulajoul et al, 2013	Performance	<ul style="list-style-type: none"> Measuring the amount of packets that can be captured and logged by a single snort IDS. 	Empirical			Recommendation to enhance the performance of IDS.	

Table 2.1-E: Summary of papers (11-13).

#	Reference id	Setting	Description of study design methodology	Type of research	Limitations	Intervention	Results	Comments
11.	Bairwa et al, 2014	Information security	<ul style="list-style-type: none"> • Introduce the most known vulnerability scanning tools. • Shows the strength of each of these tools 	Analytical	Showing the features of known vulnerability tools in easy way.		Presented an easy way to show how we can use security tools in effective ways.	
12.	Salini et al, 2015	Ontology	<ul style="list-style-type: none"> • Constructed ontology to predict and classify web attacks. • The system suggests countermeasure for the predicted attacks 	Empirical	<ul style="list-style-type: none"> • Ontology construction is done manually. • System only detects web attacks. 	Ontology is used to predict and classify the severity on each web attack predicted	Ontology Knowledge Base to classify and predict web attacks.	
13.	Elahi et al, 2009		<ul style="list-style-type: none"> • Proposed ontology to identify vulnerabilities, and the impact of these vulnerabilities on systems. 	Empirical	<ul style="list-style-type: none"> • Ontology construction is done manually. • The results of attack prediction contain a lot of false alarm. 	Ontology is used to identify vulnerabilities.	Ontology Knowledge Base for vulnerabilities identifications	

This chapter gives us an overview of researches that has been performed on IDS, in term of making them smarter and more efficient. In addition, it showed us how to implement ontologies and semantic web technologies in the field of information security as in (Kotikela, Kavi , Gomathisankaran, & Singhal, 2013) and (D. Kshirsagar & Kumar, 2013). The next chapter shows how we implement these technologies, to present our smart system.

Chapter 3

Architecture

This chapter proposes new analytical features for IDSs in DMZ zones. The analytical features were based on semantic web technologies, which is the core of knowledge-based discovery in modern systems. Furthermore, this chapter demonstrates the algorithms and the mathematical formulas that were used to build the analytical engine.

3.1 Introduction

Our proposed Smart Intrusion Detection System for DMZ specifically for DMZ zone uses ontology for creating, and displaying information about specific attacks that hit, or may hit the servers in the DMZ zone[s]. Our objectives were: to extract useful and needed security information from network traffic; to create a dynamic ontology model that serves in the DMZ needs; to create a strong analysis system that should be able to predict the incoming attacks; and to be able to predict the systems in the DMZ that may be in danger. The proposed system consists of two machines; one for traffic sniffing and packets inspection, for which we used the best known open source Network Intrusion Detection System [NIDS] snort. On the other hand, the second machine contained the analysis engine that is the smart engine, which is responsible for displaying, querying and reasoning attacks information. Figure (3.1) demonstrates the placement of our proposed smart NIDS.

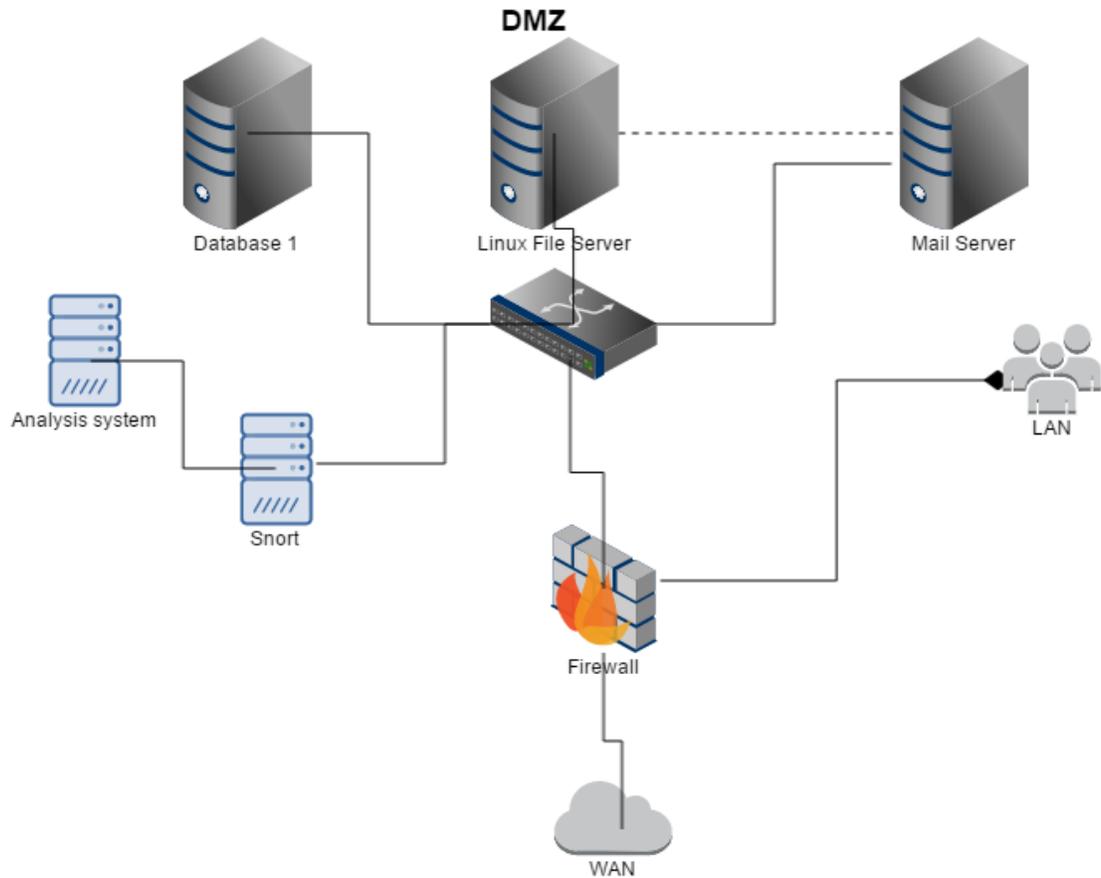


Figure 3.1: Placement of smart analysis engine.

3.2 System Components

The vulnerabilities were classified and grouped according to operating system and service type. Following subsections describe the system components and the communications between these components. Figure (3.2) shows the proposed system architecture, and its different components.

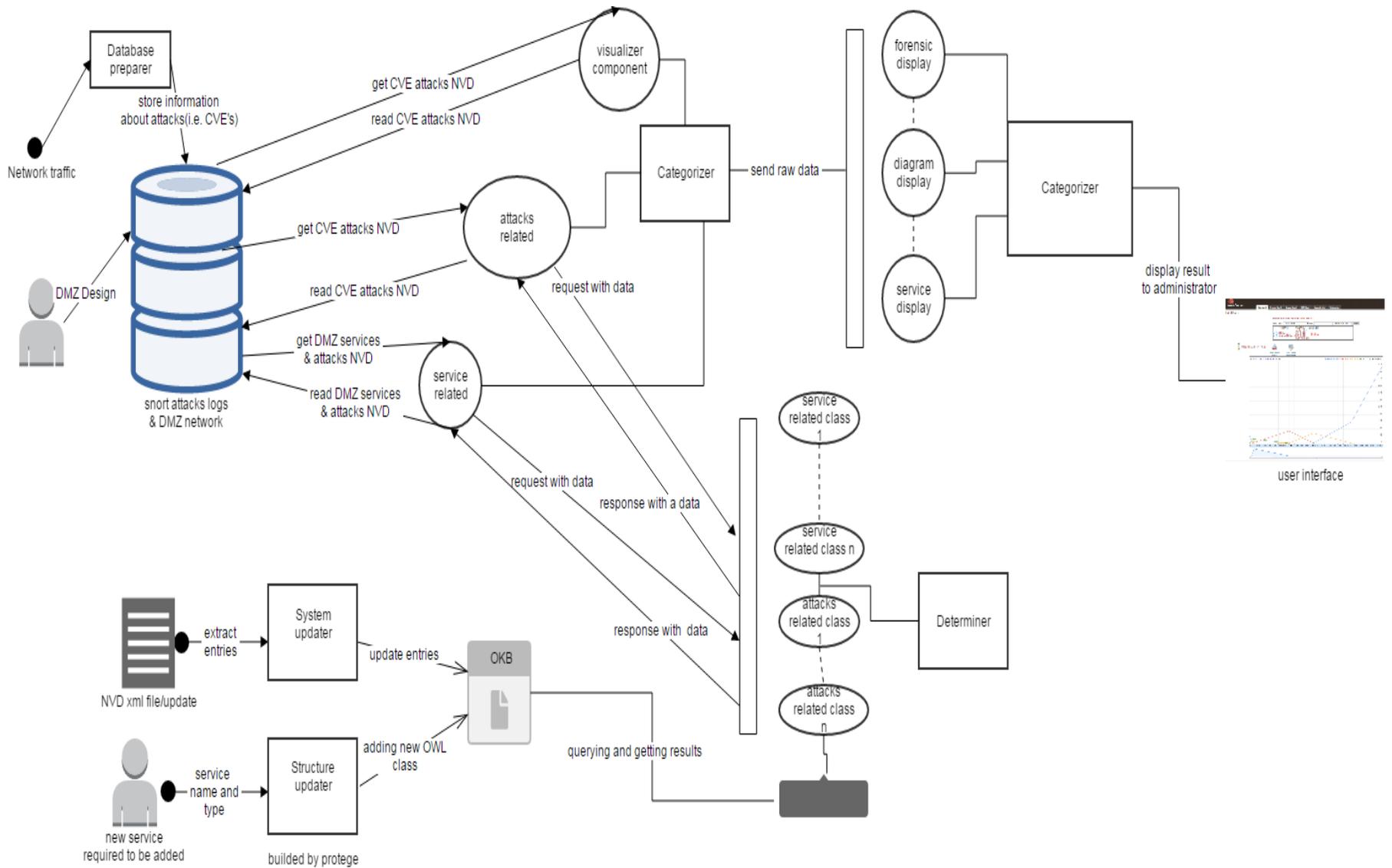


Figure 3.2: System architecture.

3.2.1. Database Preparer

The database preparer has a main function, which is storing information about attacks that hit our system, using snort log. In addition, it stores information about servers in our network. The database preparer operates by reading snort logs, applying snort rule to the read logs, converting each attack ID into NVD ID, and finally storing the new attacks IDs information in the database.

3.2.2. System Updater

The system updater is a web component. It takes the new specified NVD XML file, and adds its content [entries] to our pre-defined Ontology Knowledge Base [OKB]. NVD is an international vulnerabilities database, which contains various information about security holes existing in software's or operating systems (Nist, 2016). In this module, the new XML NVD file is uploaded to our system, and then converted into RDF triple inside our OWL file to update our created OKB system.

3.2.3. Structure Updater

The structure updater is a web module which makes changes dynamically occurring inside a pre-built ontology [T-Box part]. It will add new class to the pre-built ontology dynamically through web interface. For instance, if a specific system service is a part of our system, and it is not defined on the built ontology, this service should be added to the system dynamically without any problem using this module. This will assist the administrator to add any new service to the system dynamically, without the need for development involvement. Adding specific service class will improve and enhance the results for querying, retrieving information about specific vulnerabilities, and gives better results, since the service become a classified service, not a general service.

3.2.4. Categorizer

The categorizer is considered as coordinator in our proposed system. It is connected to various parts on the system, and it determines which data should be passed, and to what part. The categorizer consists of multiple web components [modules/methods]. Each component has its own functionality, in term of preparing, and determining the type of data that should be returned. In general, each component prepares the data to be processed and queried, then passes these data to a specific part [class] in the determiner, to do its work. After that, the determiner returned the information required. Finally the returned data is passed to the proper visualizer component, so the results of the required information will be displayed to the administrator. There are three main categorizer components. The first component is related to information about service[s]. Its function is to classify service[s] about server[s], then reading attacks from database that are related to service[s] which have been read, grouping this information, passing them to proper determiner class; to do its work by reasoning and query information from Ontology Knowledge Base [OKB], and return the results to specific visualizer component to display the results. The second component is concerned with related attacks. This component read attacks that hit specific service or system or all the systems in DMZ, then it group these attacks and passes them to specific determiner class for attacks results information. After that, the determiner passes this information to a proper visualizer method. The final component in the categorizer is the one that is connected directly to the visualizer. It reads attacks about specific service(s) or system(s) from database then passes the data directly to a specific visualizer method, to show the results.

3.2.5. Determiner

The determiner is software consisting of multiple java classes. Its main function is to work with the built OKB; in term of querying and extracting useful information's. Each categorizer component connects to one or multiple determiner class[es]. Each class is responsible for getting specific information ordered by categorizer module. After that, it groups the extracted information in a specific format, then passes the data to the visualizer module. One example of

a determiner class is a class that is concerned with related attacks information. Another example is the class concerned with grouping the predicted attacks related to an attack that might hit a service. A further example is the class that groups attacks information, and gets the services that might be related to these attacks.

3.2.6. Visualizer

The visualizer is the gate for the administrator to get the required results. It has multiple web components [modules/methods]. The main function of this module, is displaying the results in a web browser in a readable view. As well as determiner, each categorizer, or determiner component connects to one or multiple visualizer component. Each component in a visualizer is responsible for preparing the data received from the categorizer or determiner module in a specific format, calling the necessary Google visualization scripts, and visualizing the results.

3.3 Workflow process

3.3.1. System Initialization

System initialization should be performed before working on the system. It is performed in two steps. The first step is done by adding the NVD database into our built ontology [skeleton], which reflects the most DMZs in term of operating systems and services. The structure of the system services can be changed according to our needs [section 3.3.1.3 update the system structure]. NVD contains information about all security holes that are discovered, and information about these threats as XML entries in XML file (Kotikela, Kavi , Gomathisankaran, & Singhal, 2013). Each vulnerability in the NVD is identified by an ID [CVE-ID]. We extract these XML entries, and apply them to our built ontology in this initialization step, so OKB is populated. This step should be performed at least once, to be able to fill our system with some data about vulnerabilities. The second step in system

initialization is to read snort logs, then apply snort CVE rules on attacks logged, to convert SNORT-ID into CVE-ID. After that, store this information in our database. This step is a continuous process as new attacks always hit the systems. After populating the OKB and filling the database with attacks, a specific action is required by administrator, so this action is translated into a query performed on a data stored in the database by a categorizer method then the returned data from categorizer passed to determiner class to get information about these data. The determiner continuously packs the results information then passes it to a visualizer method to display the results. Figure (3.3) shows how the flow of information in our proposed system.

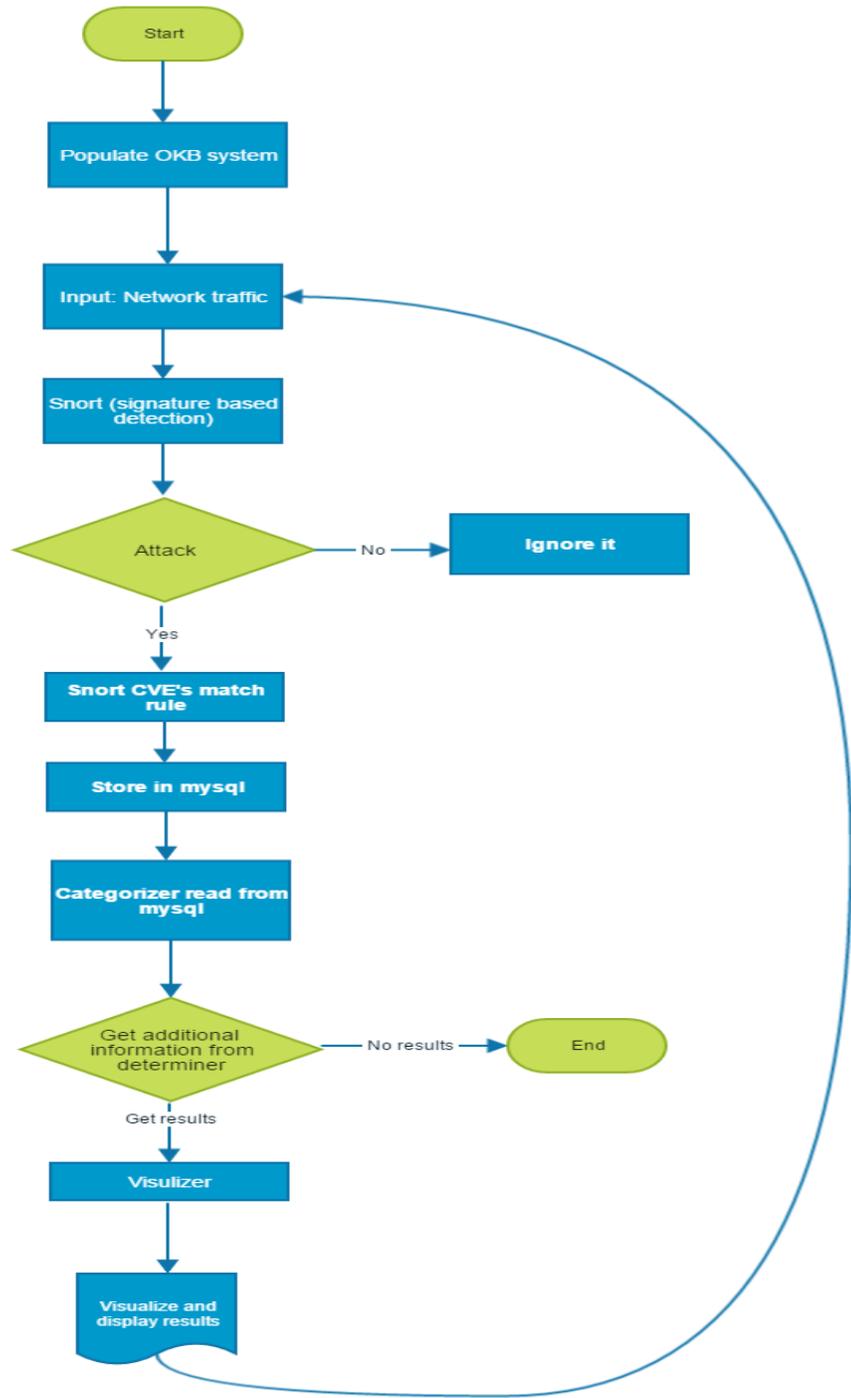


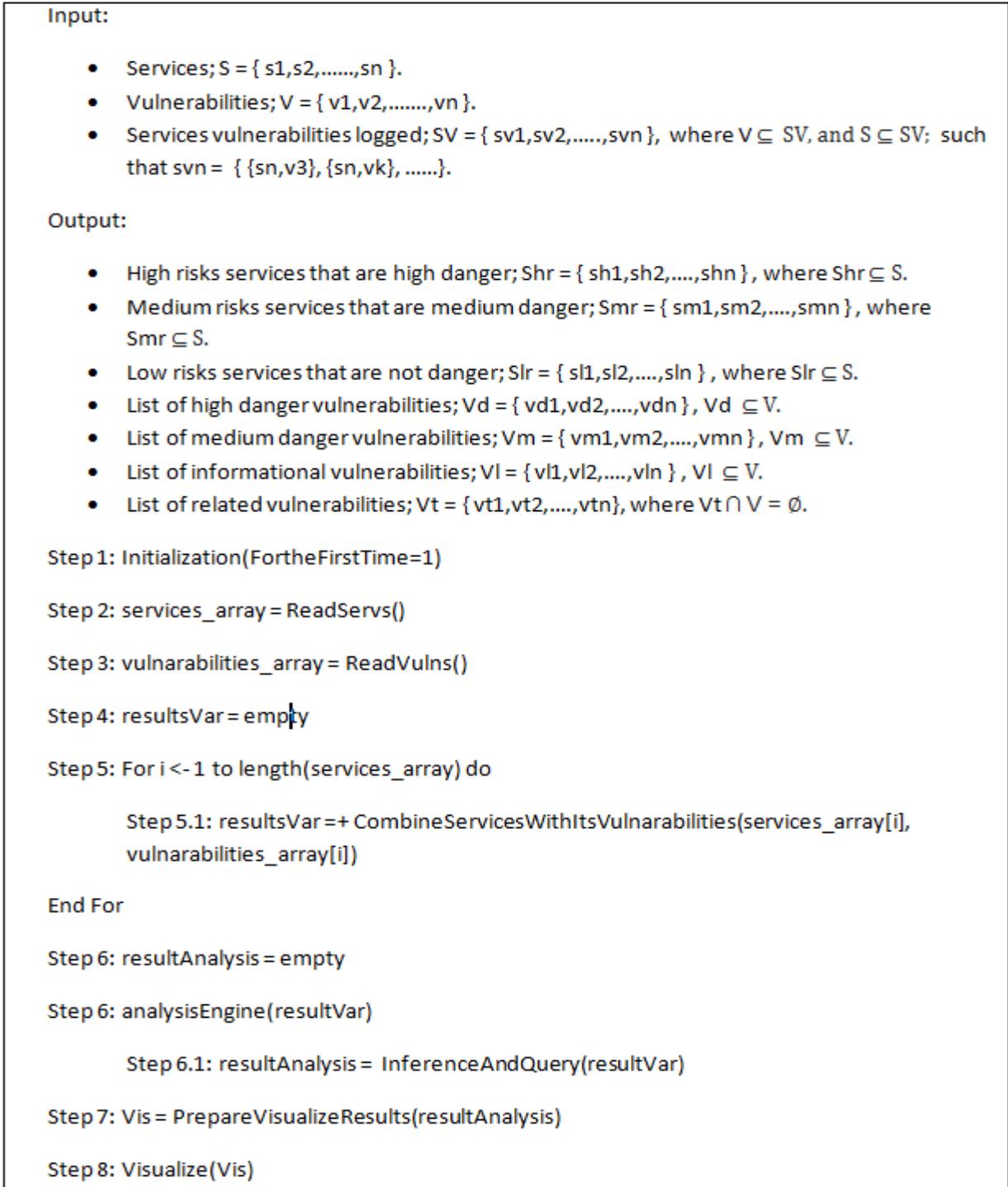
Figure 3.3: Information flow chart in the proposed system.

3.4 Algorithm

Each step in our algorithm has its own handlers and algorithm. The basic steps for our proposed semantic analysis engine for IDS are as follow:

- **Step 1:** Prepare the inference engine.
- **Step 2:** Input data from network traffic, and extract CVE's using CVE snort rule.
- **Step 3:** Read CVE's from database and pass them to inference engine.
- **Step 4:** The inference engine performs necessary data extraction from OKB.
- **Step 5:** Return the extracted data to a handler function.
- **Step 6:** Pass the data to the visualizer handlers.
- **Step 7:** Display the results.

Algorithm (3.1), shows pseudo code and mathematical formulas demonstrate the above steps.



Algorithm 3.1: Mathematical formula and pseudo code for the proposed system.

3.4.1. Prepare the Inference Engine

In this step we prepared the inference engine. The inference engine performs the necessary query, and obtains useful information about specific attacks. The preparation of the inference engine was performed using three steps: building the ontology skeleton, updating the database signature of NVD, and updating the system structure.

3.4.1.1. Building the Ontology Skeleton.

Building ontology has no specific standards; it is usually built according to our needs. We followed standards used in (Wang & Guo, OVM: An Ontology for Vulnerability Management, 2009), and added our specific classes that are related to DMZ networks, which take into accounts the standards used in NVD XML file; such as: CPE, CWE, CVE, CVSS, and other network security standards. Figure (3.4) shows the main classes in our built ontology. We mainly used protégé tool to build the skeleton that can be changed dynamically according to our needs, as we will see in later section [3.4.1.2 Update the database signature of national vulnerability database].

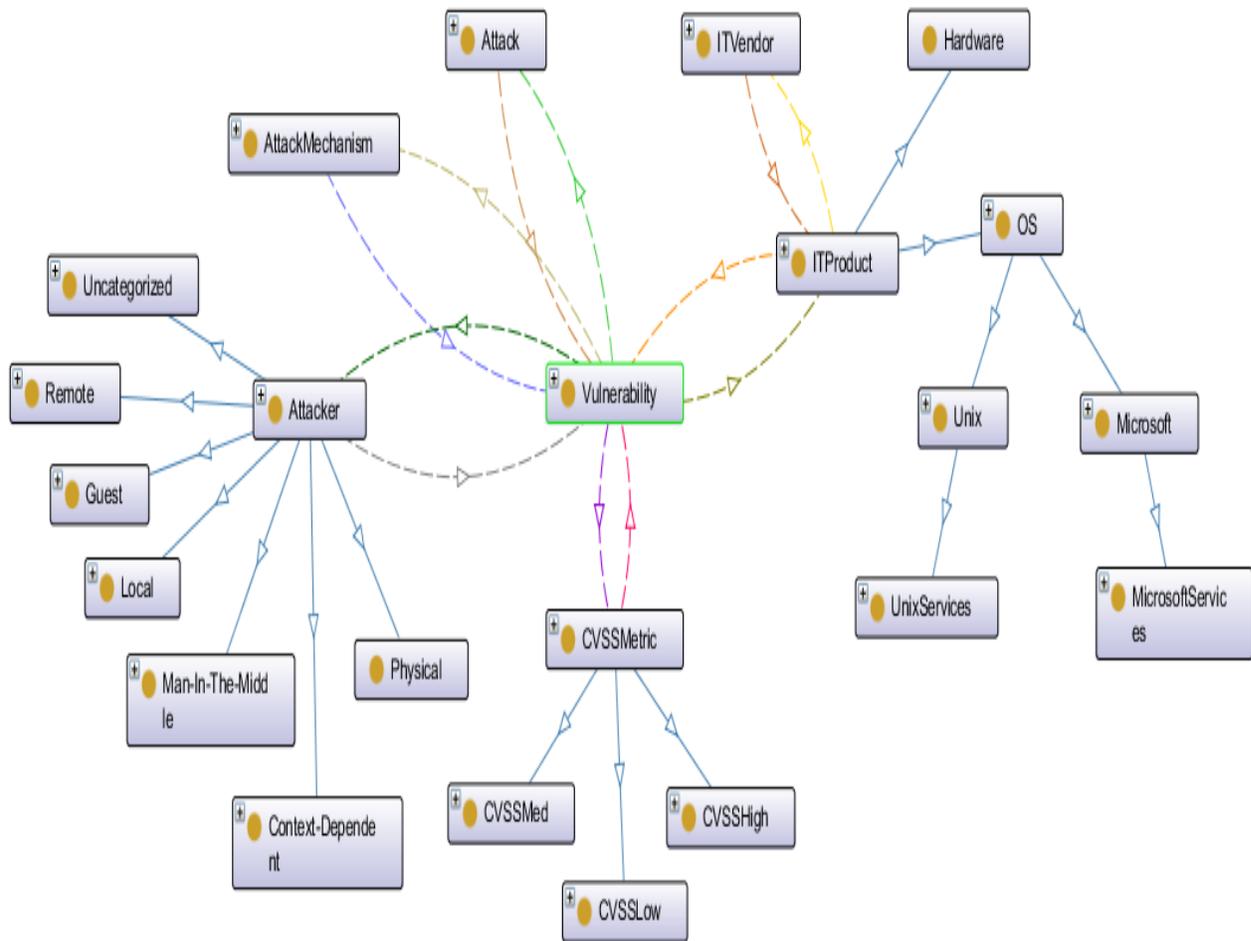


Figure 3.4: Ontology skeleton.

Our built ontology consists of the following classes:

- **Vulnerability class.** Contains a security hole that may exist in a software, or operating system, or hardware (Wang & Guo, OVM: An Ontology for Vulnerability Management, 2009). We defined the following data properties for it:
 - **hasVulnerabilityName.** Contains the CVE-ID of the vulnerability individual.

- **hasVulnerabilityData.** Contains to the data where this vulnerability individual is published.
- **hasVulnerabilityDescription.** Contains additional useful information about the vulnerability; such as: how the vulnerability can be exploited.

Vulnerability class has the following object properties, which were used to connect vulnerability individuals to others related individuals in OKB.

- **hasCVSSMetric.** Connects vulnerability individual to its CVSS score individual.
 - **hasAttackMechanism.** Connects vulnerability individual to attack mechanism individual.
 - **isAffectedITProduct.** Connects vulnerability individual to all ITProducts individuals that is affected by this specific vulnerability.
 - **hasAttack.** Connects vulnerability individual to attack individual.
 - **hasAttacker.** Connects vulnerability individual to its corresponding Attacker individual.
- **CVSSMetric class.** Contains a measurement that shows how dangerous the vulnerability is (Kotikela, Kavi , Gomathisankaran, & Singhal, 2013) (Nist, 2016). We defined three subclasses, which are:
 - **CVSSHIGH.** Contains all individuals, for which if a vulnerability individual connects to it, is considered danger vulnerability. That means this vulnerability individual could result in catastrophic damage if it is exploited, and affects the data in term of confidentiality, integrity and availability.

- **CVSSMed.** Contains all individuals, for which if a vulnerability individual connects to it is considered medium danger vulnerability. For example, it may affect availability only.
- **CVSSLow.** Contains all individuals, for which if a vulnerability individual connects to it is considered low danger vulnerability. It means that it is an informational attack only.

CVSSMetric class has the following data property:

- **hasCVSSScore.** Contains numeric value of the vulnerability dangerous rank.
- **hasCVSSIntegrityImpact.** Determines if the connected vulnerability individual with this CVSS score affected the integrity. It has three values which are: complete, meaning it affected the integrity. Partial, meaning it partially affected the integrity. None value, meaning it did not affect the integrity.
- **hasCVSSConfidentialityImpact.** Determines if the connected vulnerability individual with this CVSS score affected the confidentiality. It has three values: complete, meaning it affected the confidentiality. Partial, meaning it partially affected the confidentiality. None value, meaning it did not affect the confidentiality.
- **hasCVSSAvailabilityImpact.** Determines if the connected vulnerability individual with this CVSS score affected the availability. It has three values: complete, meaning it affected the availability. Partial, meaning it partially affected the availability. None value, meaning it did not affect the availability.

CVSSMetric has one object property, which is isCVSSMetricPartOf. This property is an inverse of object property hasCVSSMetric.

- **Attack class.** Contains the common method that is used to exploit a system (Wang & Guo, OVM: An Ontology for Vulnerability Management, 2009). We defined a data property isAttack for it, which may have one of the following common values: remote value, meaning the vulnerability individual connected to this attack, can be exploited remotely. Authenticated value, meaning that the vulnerability individual connected to this attack individual can be exploited if the attacker is part of domain users. Attack class has an object property isExploitOf, which is an inverse of hasAttack object property.

- **Attacker class.** Describes software or a human, who has a reason to compromise the computer system (Wang & Guo, OVM: An Ontology for Vulnerability Management, 2009).It has the following subclasses:
 - **Context-dependent.** Contains individuals of attackers, who have to be part of a system domain, to compromise the vulnerabilities; such as: the attacker should be authenticated to windows active directory domain.

 - **Guest.** Contains individuals of attackers who don't have to login to a system, to compromise the vulnerabilities in it. For example, guest login in a website.

 - **Local.** Contains individuals of an attacker who has to be part of a local user in a system, to compromise certain vulnerabilities in it. For instance, a local user account in Unix/Linux system.

 - **Man-in-the-middle.** Contains individuals of attackers who should be able to intercept the network traffic.

 - **Physical.** Contains individuals of attackers who have physical access to the system.

- **Remote.** Contains individuals of attackers who can remotely compromise a system. Such as: SQL injection attack against a website.
- **Uncategorized.** Contains individuals of attackers who have no certain or specific classification.

Attacker class has a data property isAttacker, which display the types of attacker in a string. This class has an object property isAttackerOf, which is an inverse of object property hasAttacker.

- **AttackerMechanism class.** Individuals from this class contain information about the mechanism used by attackers, to exploit certain vulnerabilities. This class has one data property called hasActiveLocation, which contains information about a certain vulnerability exploitation method. For instance, a certain file in JOOMLA sites is hacked by certain commands. AttackMechanism class has object property isAttackMechanismOf, which is an inverse of object property hasAttackMechanism.
- **ITProduct class.** Contains a group of products that is affected by vulnerability. It has the following subclasses:
 - **OS class.** Contains all individuals of type ITProduct, that has a vulnerability affecting the operating system or application related to a defined operating system. It has the following subclasses:
 - **Microsoft class.** Contains all individuals that affected Microsoft operating systems. It also has a subclass called Microsoftservices that contains all individuals of vulnerability products that affects Microsoft services. It has subclasses DotNet, LDAP, Exchange, IIS, NTPMicrosoft, SMB, and others.

- **Unix class.** Contains all individuals that affected Unix/Linux like operating system. It also has a subclass called Unixservices, that contains all individuals of vulnerability products that affect Unix system services. It contains subclasses Postfix, Samba, Zimbra, SSH and others.
- **OSUnknown class.** Contains all individuals that affected systems other than Unix or Windows like operating system. For example Apple IOS.
- **Software class.** Contains all products individuals of services that may exist in both Unix like systems and Microsoft systems. For instance, Apache Web Server is an application that may be installed on both Unix and Windows operating system. It has several subclasses; such as Tomcat, PHP, Oracle, Apache, and others.
- **Hardware class.** Contains all individuals of products that may be affected by physical attacks. This class is out of the scope of our research.

ITProduct class has an isITProduct data property which contains a literal of all products affected by certain vulnerability. It has the following object property:

- **isAffectedITProduct object property.** This is an inverse of hasVulnerability object property.
- **isVendorOf object property.** This is used to links ITProduct individual to ITVendor individual class.
- **ITVendor class.** Contains information about the supplier of ITProduct individual (Wang & Guo, OVM: An Ontology for Vulnerability Management, 2009). It has a data property isAvendor, which contains a string of information about the supplier

of a specific ITProduct individual. It has an object property hasVendor that is an inverse of isVendorOf object property.

3.4.1.2. Update the Database Signature of NVD

NVD is an XML format file; it is updated frequently by NIST organization. In order to be able to query and inference the vulnerabilities that hit our systems, we converted these XML files into RDF/OWL triples that match our built ontology. Figure (3.5) shows how the XML file was added to our ontology. Algorithm (3.2) shows pseudo code and mathematical formulas that demonstrates how an updating database signature was performed.

Input:

- XML entries, $E = \{ e_1, e_2, \dots, e_n \}$, where $e_n = \{ \text{vulnerability}_n, \text{products}, \text{CVSS metrics}, \text{summary} \}$.

Output:

- RDF triples classified into services classes $S, S = \{ s_1, s_2, \dots, s_n \}$ such that s_n represent service name, and $s = \{ \text{vulnInfo1}, \text{vulnInfo2}, \dots, \text{vulnInfon} \}$. Also $\text{vulnInfo} = \{ \text{vulnerability}, \text{attack}, \text{attacker}, \text{attackerMechanism}, \text{vendor} \}$. In conclusion vulnerabilities should be classified according to service name, so that $\text{vulnInfo} \subseteq s \subseteq S$.

Step 1: foreach entries in NVD xml file

Step 2: if entry is not exist before in our OWL file then

Step 2.1: $\text{attacker}, \text{attack}, \text{attackMechnism} = \text{ExtractattackInfoMethod}(\text{entry})$.

Step 2.2: $\text{CVSSMetric} = \text{ExtractCVSSMethod}(\text{entry})$.

Step 2.3: $\text{ITProduct}, \text{Vendor} = \text{ExtractProductsMethod}(\text{entry})$.

Step 2.4: $\text{CVSSMetric} = \text{ExtractCVSSMethod}(\text{entry})$.

Step 2.5: $\text{Vulnerability} = \text{ExtractVulnerabilityMethod}(\text{entry})$.

Step 2.6: $\text{SaveToRDF}(\text{Vulnerability}, \text{attacker}, \text{attack}, \text{attackMechnism}, \text{ITProduct}, \text{Vendor}, \text{CVSSMetric})$

End If

End For

Step 3: Commit Changes to File

Algorithm 3.2: mathematical formula and pseudo code for updating system vulnerabilities signature.

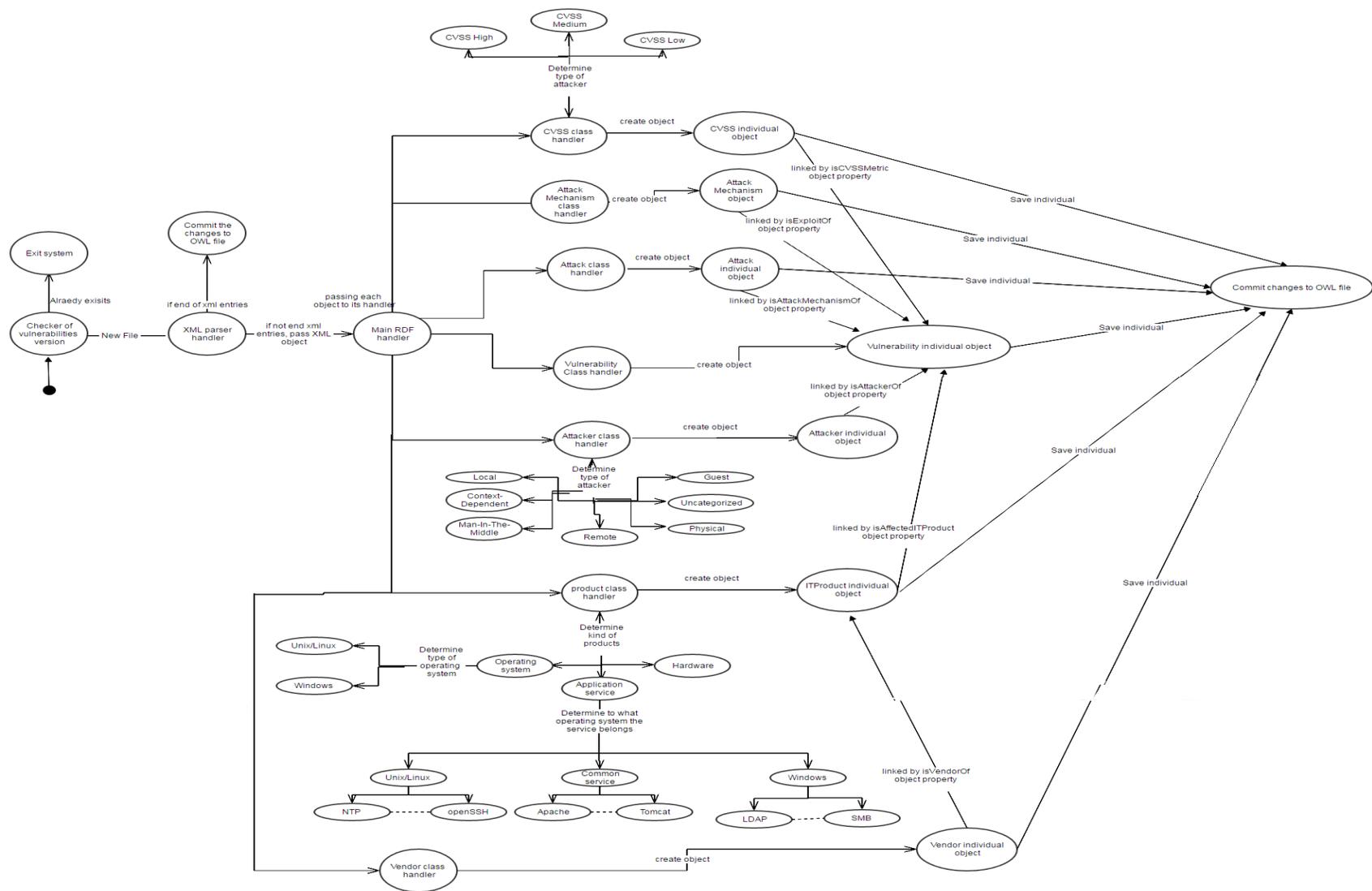


Figure 3.5: Adding XML entries into built ontology.

3.4.1.2.1. Detailed Algorithm

The NVD database is continuously updated, so the database signature updater is a continuous process. The implementation of this module was performed using four major steps. These steps were implemented using PHP scripting language, specifically using simpleXML library for data extracting from XML feed, and easyRDF library to deals with RDF triples and OWL files. These steps can be summarized as follow:

- Read NVD xml file then parse each entry in it.
- If the entry does not already exists in our built OKB, then:
 - The placement and the conversion of Vulnerability and ITVendor are performed normally, by calling a handler method to each type. The handlers will extract the objects from the the XML file then convert them in to RDF triple.
 - The extraction of CVSS score is performed by calling CVSS handler method. This handler checks the type of CVSSscore according to its score value. After that, it places an entry individual according to that score, then links this individual to the vulnerability individual using object property hasCVSSMetric. The value of the CVSS score is determined according to the following three values; CVSSHIGH, in the case if the score value of the attack is greater than 7.0. CVSSMed, if the score value of vulnerability in rang of 4.0-6.9, and finally, CVSSLow if score value between ranges 0 - 3.9.
 - The placement of Attackmechanism, Attacker, and Attack individuals are done by applying lexical analysis code, to summary part for each XML entries in NVD file. So the information extracted about the attack mechanism, attacker,

and attack are linked to vulnerability individual by hasAttackMechanism, hasAttacker and hasAttack accordingly.

- Placement of products is performed by calling the product handler method, in which the products are categorized according to operating system and application type that were built by protégé, and can be can be changed and edited dynamically as we will see later [section 3.4.1.3 Update the system structure]. For example, let's say that a vulnerability CVE-123 that may hit an Apache web server, and it can also hit an IIS web server. In this situation, the product handler method should place two individuals in the product class; one under Apache class, and the other one under IIS class. It should then link each individual of those classes with object property hasVulnerability to CVE-123.
 - The placement of ITVendor individual is performed by calling the ITVendor handler. After that, the handler links the extracted individual to ITProduct individual using hasITVendor object property.
- Publish the RDF graph, update the OKB, and commit the changes to OWL file.

3.4.1.3. Update the System Structure

In this module of our proposed analysis engine for intrusion detection engine, we allowed the network administrator to update the skeleton of the ontology [T-Box] dynamically. We created this step to allow the system to be flexible, according to the changes that occurs in an organization. This means that the organization does not have to call the developer every time a change occurs in its infrastructure. Furthermore, given that technologies always changes, and the need for new services always appears, it does not make any sense to call a developer, every time an organization needs to add new service to its infrastructure. We built the skeleton of ontology based on well-known services that are common and exist on the most DMZs. But each organization has its own needs, and for that we added this feature to allow organization

to add its required services, to provide better query and reasoning about the attack that may have hit the added service, or any other related services. Since individuals of unclassified service or a service that does not belong to a certain class, go to a general class called Software class providing inferior results, when administrator requires information about that service. On the other hand, applying this step will allow the whole structure to be changed. Thus, it requires erasing all the NVD XML, and re-entering them, to allow the entries to be fitted to match the new ontology structure. Figure (3.6) demonstrates the mechanism used in this module. Algorithm (3.3) shows the mathematical formulas, and pseudo code for this module.

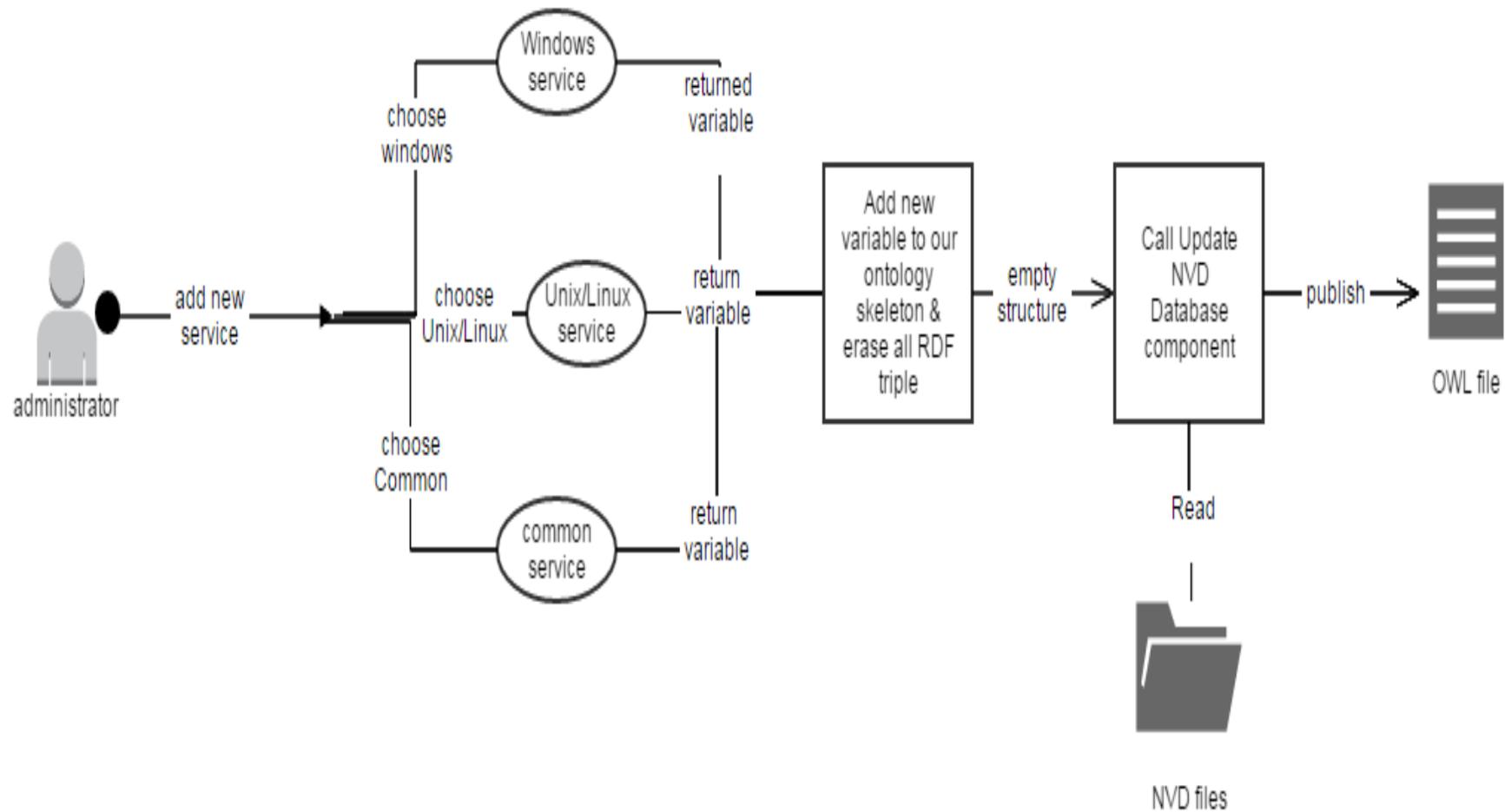


Figure 3.6: Mechanism in structure update.

Input:

- New service (s_m).

Output:

- OWL file containing list of services S , such that $S = \{ s_1, s_2, \dots, s_n \}$ where $s_m \in S$.

Step 1: `ServiceName = GetService()`

Step 2: `ListOfServices = ReadServices(OWLFile)`

Step 3: `AddService(ListOfServices,ServiceName)`

Step 4: `DeleteFile(OWLFile)`

Step 5: `ReconstructFile(OWLFile)`

Step 6: `AddToFile(OWLFile,ListOfServices)`

Step 7: `UpdateNVDDatabase()`

Step 8: `CloseFile(OWLFile)`

Algorithm 3.3: Structure updates pseudo code and mathematical formula.

We scripted this module using PHP scripting language, and easyRDF to deal with OWL files. The following steps are followed to complete this module:

- Administer the request to add new service.

In this step we allow the administrator to choose from three options to add new service in the created ontology. The first choice is adding a Windows service. The second choice is for if the request requires adding Unix/Linux service. The last option, if it is a common service, which may work on both Unix/Linux and Windows operating systems.

- Open the structure OWL file for editing.

According to the choice made by the administrator in the first step, we add necessary headers to the ontology structure OWL file.

- Erase all RDF entries from our OKB file.
- Copy new structure from structure OWL file to our new OKB file.
- Re-enter all the NVD xml files, and updates the OKB file, using the update NVD database module for all saved NVD xml files.

3.4.1.4. Input Data from Network Traffic and Extract CVE's

Traffic that passes through The DMZ normally is an active continuous process. Snort, which is an open source network intrusion detection system, plays a critical role in our proposed system. It sniffs network traffic and log security attacks into a database. In conclusion of this process, each attack which occurred has an ID given to it by snort. In the proposed system we take a security logs from snort log, then apply snort CVE's rule to it, which is a PHP script that converts snort ID to CVE's ID, so that we can extract additional information's about attacks in the analysis engine, figure (3.7) demonstrates this step. The following algorithm (3.4) explains the procedure performed in this module.

Input:

- Snort logs, $SL = \{ sl1, sl2, \dots, sln \}$, where $sln = \{ \text{Attack-Snort-Id, port_number, dateofAttack, } \dots \}$.

Output:

- CVE logs, $CL = \{ cl1, cl2, \dots, cln \}$, where $cln = \{ \text{CVE-Id, IP-Attcker, IP_Victium, port_number, dateofAttack, } \dots \}$.

Step 1: While System is Alive

Step 2: SnortLog = GetLog()

Step 3: CveLog = ConverLogAnDApplyCVE-Rule(SnortLog)

Step 4: StoreCveLog(CveLog)

End While

Algorithm 3.4: Converting Snort log to CVE log pseudo code.

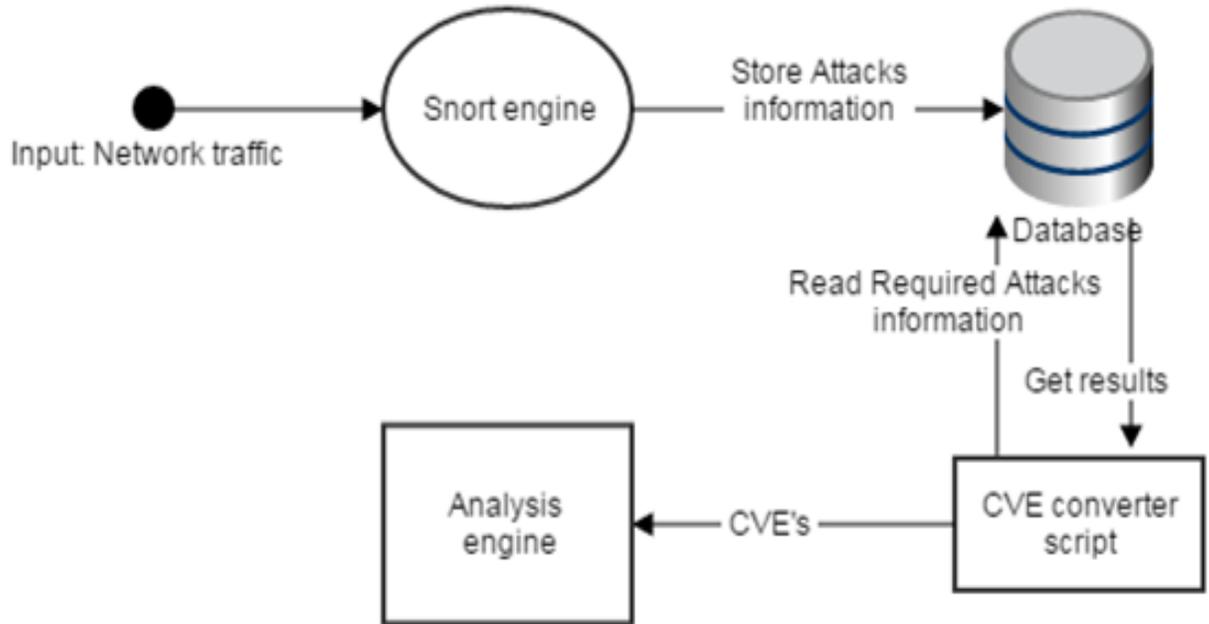


Figure 3.7: Input Data from Network Traffic and Extract CVE's.

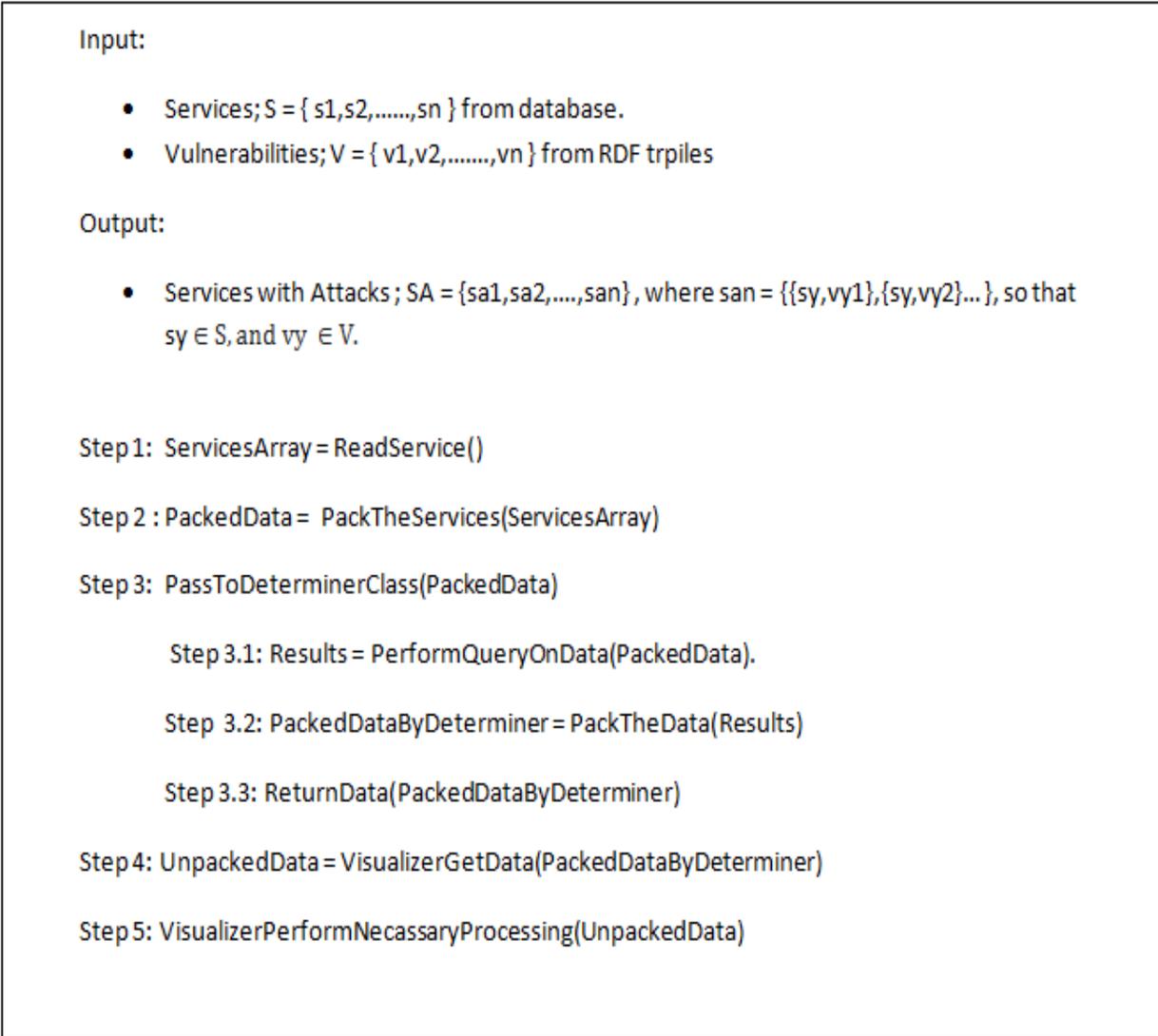
3.4.1.5. Reading CVE's and preparing the Data

The categorizer in our proposed system is responsible for reading CVE's and other information from the database. In addition, it is responsible for categorizing these data and passing them to the determiner or the visualizer -step 3 and step 5 in the algorithm [section 3.4.1.3.1]. All categorizer methods were written in PHP scripting language. Figure 4.2 shows how a categorizer module works. As mentioned before [section 3.2.4], there are three main categorizer methods, which are methods that concerned with services, methods that concerned about related attacks, and methods that connect directly to the visualizer. Each method is invoked according to the type of data required by the administrator, to be displayed and visualized.

3.4.1.5.1. Methods that Concerned About the Services

These methods are concerned with displaying information about attacks that hit a single service or group of services, in one system or multiple systems. These methods can be categorized into three subcategories. The first methods category is concerned with all system services in the whole DMZ. The second category is concerned with attacks that may be used to hit a single service. The third group is the same as second group, but for multiple services. This categorization was performed, because each category has its own way of preparing data read from the database. Furthermore, each method has its own protocol, for sending data to the determiner class.

The following algorithm (3.5) is used by methods that are concerned about all the services in the DMZ from the begging of getting data to display results.



Algorithm 3.5: Pseudo code used by Methods that Concerned About the Services.

In the first two steps in our algorithm, we normally read system services and pack them using as followed format; `service1#service2#.....servicen`.

The second and the third subcategories are the same as first one. Conversely, they are different in the way they communicate with its cross bonding determiner classes, as we mentioned early in this section. For example, the following protocol is used to communicate if we want to keep track of a server, to determine which service belongs to what server: `service1@IPn#servicen@IP1#servicen@Pn+1`.

3.4.1.5.2. Methods that Concerned About the Related Attacks

These methods use almost the same steps algorithm used in methods related to services. But here we have additional two steps, which are reading attacks from snort database, and keeping track of these attacks. So the modified algorithm 3.6 will be as follows:

Input:

- Services ; $S = \{ s1, s2, \dots, sn \}$, and internet addresses $IP = \{ ip1, ip2, \dots, ipn \}$, from database.
- Services and IPs addresses; $SIP = \{ sip1, sip2, \dots, sipn \}$, where $sipn = \{ sk, ipk \}$, where $sk \in S$, and $ipk \in IP$.
- Vulnerabilities; $V = \{ v1, v2, \dots, vn \}$ from RDF trpiles

Output:

- Services IPs with Attacks ; $SIPA = \{ sipa1, sipa2, \dots, sipan \}$, where $san = \{ \{ sipy, vy1 \}, \{ sipy, vy2 \} \dots \}$, so that $sipy \in SIP$, and $vy \in V$.

Step 1: `ServicesArray = ReadService()`

Step 2: `IPS = ReadIPs()`

Step 3: `ServicesIPSArray = CombineServiceWithIP(ServicesArray,IPS)`

Step 3 : `PackedData = PackTheServices(ServicesIPSArray)`

Step 4: `PassToDeterminerClass(PackedData)`

Step 4.1: `Results = PerformQueryOnData(PackedData).`

Step 4.2: `PackedDataByDeterminer = PackTheData(Results)`

Step 4.3: `ReturnData(PackedDataByDeterminer)`

Step 5: `UnpackedData = VisualizerGetData(PackedDataByDeterminer)`

Step 6: `VisualizerPerformNecessaryProcessing(UnpackedData)`

Algorithm 3.6: Pseudo code used by Methods that Concerned About the Related Attacks.

In step two in the algorithm, we read attacks that hit a particular system service, or a group of systems services from snort database. After that, we group these data, which are about attacks with data about systems services in a predefined format then sending the packed data to a determiner class. These methods were categorized according to number of systems services, since we want to get information about attacks that hit services, or getting the ranks of the attacks that hit these systems services. On the other hand, all of these methods followed the same sequences, but with some differences in the parameters, and protocols used in sending data to the determiner class. The protocol used in sending the data is;

```
service1@IPn<cve1,cve2 ..... cven>#servicen@IP1<cve1,cve2 .....  
cven>#servicen@Pn+1<cve1,cve2 ..... cven>.
```

3.4.1.5.3. Methods that Connected Directly to the Visualizer

These methods are considered the simplest. They require neither grouping for data, nor creating a protocol of passing data to the determiner class, since it does not call any determiner class. These methods make a normal query to data from snort database to get information about attacks. After that, it passes the requested data directly to the required visualizer method, to display the results.

3.4.1.6. Querying and Inference Necessary Information from OKB

The determiner is the system module responsible for dealing with OKB, in terms of query and getting necessary information. It consists of multiple java classes, each with its own functionality in terms of unpacking the data taken from categorizer, querying the data from OKB, and packing the results back to the visualizer. These classes use Jena library, a set of java classes that are programmed to deal with semantic web technology. It also has a group of APIs to read, query and inference OKB. In addition, this library has a built-in capability that allows easy interaction with a different type of reasoner, such as PELLET, HAMLIT, and others (Semantic Web, 2012) (Apache Jena, 2015). As mentioned before, each categorizer method has one or more cross-ponding determiner class. For instance, service method [section

3.4.1.5] has multiple sub categorizes; each subcategory connects to only one class from the determiner. In this module we also categorized the determiner classes into two subcategories; one category contains all service related classes, and the other category is attacks related classes.

3.4.1.6.1. Classes Related to Services

In this group of classes, the types of information that will be queried are the information's about attacks that might hit a system service, or group of systems services. The main steps used in these classes can be summarized as follows:

- **Step 1:** Read the data passed from the categorizer method.
- **Step 2:** Extract these data into variables.
- **Step 3:** Build the necessities query, using SPARQL and different reasoners.
- **Step 4:** Pack the data in a special format.
- **Step 5:** Pass the data back to visualizer method.

Each determiner class gets the data from categorizer method in a special format. The first thing which should be performed is to unpack these data as we mentioned in earlier sections, each categorizer packing the data in a special format then passing the packed data to a determiner class. The next steps should be reading it and querying it. Finally, it must use its own packing method, to prepare the data to be returned for the visualized method. For example, the query related java class reads the systems services passed to it, and then builds queries to get information about the attacks that might hit these services. After that, the information will be returned to the visualizer method in the following format: \$ListOfReslated[] = serv1, \$ListOfReslated[]= \$cve2, and so on.

Another example is if we want to display the related attacks that might hit a service or a group of services. It uses almost the same procedures used in related system services. But here the communication protocol in returning the information data is different. The protocol used in this class is as follow: Service1[] = CVE1#ProductsHit, Service1[] = CVE2#ProductsHit Servicen[] = CVEn#ProductsHit.

3.4.1.6.2. Classes Related to Attacks

These types of classes are concerned with returning information about attacks that are related to an attack, or getting the CVSS metrics of attacks. Classes from this category, followed the same steps used by classes related to the services, but, with changes in type of query and inclusion of CVSS score. In addition, the protocols used in these classes are quite different. One example is the class that is concerned with getting all the attacks that might hit our systems, with the rank of each attack shown in the result. The following protocol is used in this class, to pass the result information back to the visualizer.

ServiceName1CVSSType1[] = cve1 , ServiceName1CVSSType1[]= cve2,,
ServiceNamenCVSSTypen[]=cven.

3.4.1.7. Prepare Data to be Displayed and Visualized

The visualizer module is responsible for displaying the results in a nice GUI. It accepts the data that are passed by the determiner class. It is consist of multiple web methods that were written in PHP scripting language. Each method uses a different Google visualization library to display the results in different views. Google visualization library is a set of scripts that are written by Google Co. to display data in a nice visualized view. Each library has its own data format, which should be followed to visualize the results. We followed Google's guide to prepare and display our data results (Google org, 2015). The following subsections demonstrate some of the visualization methods that were used in our proposed system.

3.4.1.7.1. Pie Chart Method

We used these kinds of charts to display information about the attacks that might hit systems services, and the attacks that already hit systems services. We accepted the data from the determiner methods, specifically from the methods that are related to the services, prepare the information accepted, and finally display the results.

3.4.1.7.2. Annotation Chart Method

This method displays the data in a time line view. We used this type of chart to display attacks statistics that occurred before a specific date time. In other words, this method displays historical information about attacks. This method accepts data from the attack related determiner method then performs necessary preparation to the data before displaying it.

3.4.1.7.3. Word Trees Chart Method

This method displays the result in a word trees format. We used this method to display related systems services that might be in danger in our DMZ. This method accepts its data from an attacks related service determiner methods.

3.4.1.7.4. Column Chart Method

This method is the same as the pie chart method. However, it has minimal changes in data preparation.

3.4.1.7.5. Organization Chart Method

This method takes its data directly from the categorizer method. For that, it is considered the fastest method that displays result for the administrator.

3.5 Summary

The goal of this chapter is to show how the semantic analytical engine for IDS was developed. Also, we showed the procedures followed to make the system semantically analyze each attack logged, so it can perform prediction about incoming attacks or services that might be in danger. In addition, we explained how semantic connection between IDSs and NVDs was made.

This chapter shows the steps needed to be performed on each attack logged, before visualizing the results.

Furthermore, this chapter demonstrates the algorithms and mathematical formulas that we used to build the analytical engine. In addition, we showed that the predictions are based on predefined criteria's that are passed, to be queried.

Then next chapter shows how we validated our system using real data (from KDDCup99 dataset). Furthermore, explain how we measured the performance of our system using a queening model, and Anylogic simulation tool.

Chapter 4

System Validation

In the previous chapter we showed how we built our prototype using PHP scripting language, and Jena java library. In this chapter we will show how our prototype was validated using three different techniques. The first technique is by validating our system using KDDCup99 dataset. KDD is considered the main source of attacks that hit the DMZ system. On the other hand, the source of vulnerability attacks information is the NVD database. We used the NVD database from 2002 to 2008 in the implementation of our experiments, which it's stored in the OKB as discussed earlier [chapter 3]. The second technique is the queuing model, which we used to validate our system in a heavy multi-user environment. The Anylogic simulator was the last technique we used to validate the system. Furthermore, in this chapter we present some of the web user pages that are shown to the administrator.

4.1 KDDCup99 Experimental Data

KDDCup99 is the most widely used data for evaluating and validating the intrusion detection systems. In our experiments, we used these data; to predict attacks related to attacks stored in this dataset. The Defense Advanced Research projects Agency [DARPA] and Air force Research Laboratory [AFRL], MIT Lincoln Laboratory had collected these large datasets from real networks, for the evaluation of computer network intrusion detection systems (Panda & Patra, 2007).

We used and queried the data in this dataset, to predict new and related attacks from the created OKB. We performed different tests on these data to get predicted attacks based on the tests requested. These experiments were performed on an Intel dual core 3.0 GH machine, with 3 GB memory, and a PHP java support environment. The tested machine has the following services enabled on it; snmp, internet_information_server, frontpage and pdg_shopping_carte [in CPE format].

4.1.1. Measuring the Accuracy of attack prediction

The main source of attacks predictions is our created OKB; we use it to predict the attacks related to the stored sniffed attacks, by using ontology inferences reasoners and SPARQL queries. The predictions are based on the reasoners to extracted additional facts about specific attacks; we depend on four metrics passed to reasoners to extract information which are: CVSSScore, AttackMechanism, Attacker, and products. We performed three tests and manipulate the threshold of these parameters.

4.1.1.1. Predictions based on two parameters

In this test we pass four parameters [which are mentioned earlier], and we limit the threshold of the resoners to two, which means that if two parameters that are passed or more match in the OKB, it return vulnerability as predicted attack; i.e. if for example vulnerability CVE-123 that are sniffed and logged has a CVSSScore is high, Attacker type is remote, AttackMechanism is by .htaccess file, and the vulnerable product is Apache server 1.3. The reasoners will check the OKB files if two or more of the above information is true for the vulnerabilities inside it, For instance if CVE-456 inside OKB affected Apache Server 1.3, and has CVSScore high the resoner will ignore the Attacker type and Attack Mechanism and return that CVE-456 is predicted attack for CVE-123. Table (4.1) shows the results of prediction for services if we limit the threshold of reasoners to two.

Table 4.1: Two parameters metric for accuracy prediction

Service name	Predicted	True alarm	False alarm	Accuracy percentage
snmp	17	4	13	23.5%
internet_information_server	11	2	9	22.3%
frontpage	45	31	14	68.8%
pdg_shopping_carte	3	2	1	66.6%

4.1.1.2. Predictions based on three parameters

We repeat the same experiment as in [section 4.1.1.1]. But in this experiment we made the threshold of reasoners to three instead of two. In other word, must at least three of passed parameter be true to return an attack as predicted attack. Table (4.2) shows the result of prediction based on three parameters.

Table 4.2: Three parameters metric for accuracy prediction

Service name	Predicted	True alarm	False alarm	Accuracy percentage
snmp	9	4	5	44.4%
internet_information_server	2	2	0	100%
frontpage	39	31	8	79.5%
pdg_shopping_carte	2	2	0	100%

4.1.1.3. Predictions based on four parameters

We repeat the same experiment as in [sections 4.1.1.1 and 4.1.1.2]. On the contrary, in this experiment we made the threshold of reasoners to four, i.e. the four passed parameters must be true to return an attack as predicted attacks. Table (4.3) shows the result of prediction based on four parameters. The only misses that occurs in frontpage service for example is because the value of attacker type does not exists in original NVD XML file that was converted into RDF triple [the summary part of the original XML file does not contains information about attacker], so that the reasoners added a virtual value from the passed attacker attribute value and made it the value that is missed.

Table 4.3: Four parameters metric for accuracy prediction.

Service name	Predicted	True alarm	False alarm	Accuracy percentage
snmp	5	4	1	80%
internet_information_server	2	2	0	100%
frontpage	34	31	3	91.2%
pdg_shopping_carte	2	2	0	100%

From the results of three experiments we can see if we increase the threshold of reasoners, the prediction accuracy increased. But, the attacks relation between different services will be decreased, thus if we concerned about attacks relation we can safely increase the threshold of the reasoners.

4.1.1.4. Comparing the results with other systems

We compare the results of inferences made by reasoners [reasnoer threshold = 4] used in our system with two other systems that are used reasoners for attacks prediction (Salini & Shenbagam, 2015) and (Elahi, Yu, & Zannone, 2009) , also we compare our approach with other system that uses other datamining algorithms [GA] (Hoque, Mukit, & Bikas, 2012) for attacks prediction. Table (4.4) shows the result of comparisons.

Table 4.4: System comparisons.

System Used	Prediction Accuracy
Our System	92.8%
Other OKB infernces 1	84.6%
Other OKB infernces 2	70.5%
Using GA	53%

As we can see from the results our system gives the best results because the prediction depends on two sources of data semantically connected for attack prediction, not as other systems that use pre trained data to do prediction, nor uses pre built ontology to predict attacks from normal network traffic. Thus, this makes the system more practical to be applied for production environment.

4.2 Queuing Model

To validate our proposed system in other way, rather than using KDDCup99 benchmark, we considered modeling our system using a queuing model. Queuing model is a mathematical model (Wikipedia, 2016) (Belch, Greiner, de Meer, & Trivedi, Queueing Networks and Markov Chains, 1998), which we used in our proposed system to predict waiting time and how much time each request is takes to be processed, by different modules of the system.

As mentioned earlier we have three modules, so we must have three different queues: categorizer, determiner and visualizer queues. Each request [query requested by administrator], comes to the system in the form of 2000 rows per query [i.e. we limit the query to only 2000 rows per request, which we found good in term of time required for processing. Table (4.5) demonstrates different chunk sizes and the time required by different part to process it, the time taken by determiner was unchanged, because, the data set contains replicated attacks extracted and filtered before entered this module].

Table 4.5: Time Required to process different sizes of requests at same time.

Number of attacks per request	Time in categorizer(Sec)	Time in determiner(Sec)	Time in visualizer(Sec)	Total time(Sec)	Number of attacks per request
500	3.82	1.4	1.014	6.234	500
1000	6.6	1.4	1.033	9.033	1000
2000	13.8	1.4	1.064	16.264	2000
4000	25.9	1.4	1.13	28.43	4000
6000	37.8	1.4	1.22	40.42	6000
8000	56.7	1.4	1.32	59.42	8000

These requests enter the categorizer first and are grouped by it, then mostly go to determiner, and finally to visualizer to display results. However, in some cases these grouped requests go to the visualizer directly from categorizer, to display the information grouped. So each request consists of a chunk of rows [2000 rows], and it is treated as one request by the system. These requests come to our system in a poisson distribution in average, with one request each second; since we know that in each network there is a peak time and normal and we care about events that are arrive. In our system, we can use Jackson open network model, to present our system, because the system consists of multiple queues. In addition, the requests come from outside the system. Figure (4.1) below shows the queuing model for our proposed system.

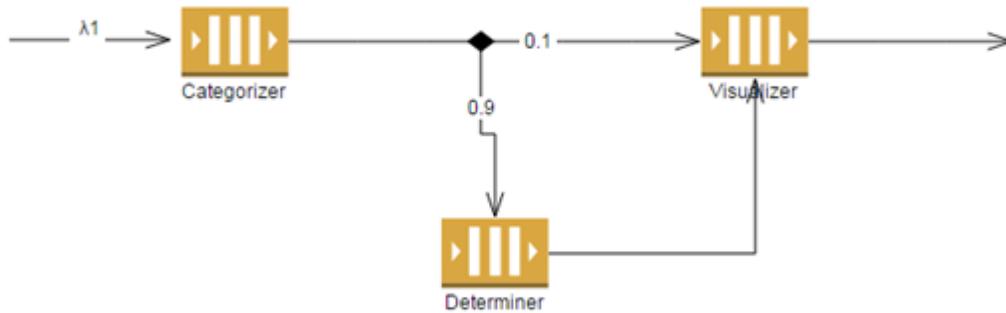


Figure 4.1: System queue model.

So, we have three queues; each queue in our system is an $m|m|1|\infty$ queue, since the requests come to our system in a poison distribution, with an average of one request per second. We have one server in each queue, to serve the requests. In addition, each queue type is a FIFO and we have infinite queue size for each queue in the system. The following information represents data obtained by performing sample query on KDD data set, and it records how much time each component require to perform the necessary data processing. From these data we could predict the behavior of the proposed system, so that we can estimate how much each requests required to be processed. Furthermore, the throughput of a busy system, the utilization of the system, the probability of the system being idle, and other useful information shows us how the system will reacts in a busy environment. In the following calculations: $q1$ represent categorizer, $q2$ represent determiner and $q3$ represent visualizer.

$$\lambda_1 = 1/\text{seconds} [\text{chunk of 2000}] [\text{rate of arrival for } q1], \mu_1 = 13.8 \text{ sec} [\text{service time for } q1]$$

$$\lambda_2 = 0.9 * 1 = 0.9 [\text{rate of arrival for } q2], \mu_2 = 1.4 \text{ sec} [\text{service time for } q2]$$

$$\lambda_3 = 0.1 * 1 + 0.9 = 1 [\text{rate of arrival for } q3], \mu_3 = 1.064 \text{ sec} [\text{service time for } q3]$$

- **Total throughput** = 2.9 requests.

- **Expected number of requests in the system (L)**

By using the normalization condition we can see that this geometric sum is convergent, iff $\lambda/\mu < 1 = P$, Thus:-

$$P(q1) = 0.07 (\lambda1/\mu1) \text{ for queue1.}$$

$$P(q2) = 0.64 (\lambda2/\mu2) \text{ for queue2.}$$

$$P(q3) = 0.9 (\lambda3/\mu3) \text{ for queue3.}$$

$$L(q1) = 0.075 \text{ (expected number of requests in queue q1).}$$

$$L(q2) = 1.77 \text{ (expected number of requests in queue q2).}$$

$$L(q3) = 9 \text{ (expected number of requests in queue q3).}$$

$$L = 0.075 + 1.77 + 9 = 10.8 \text{ requests is the Expected number of requests in the system.}$$

- **Waiting time for the next request to be served (W)**

$$w(q1) = 0.075 \text{ second in q1.}$$

$$w(q2) = 1.96 \text{ second in q2.}$$

$$w(q3) = 9 \text{ second in q3.}$$

$$W = 0.075 + 1.96 + 9 = 11.04 \text{ second Waiting time for the next request to be served.}$$

- **Probability of the system being idle**

$$p(0,0,0) = 0.93 * 0.36 * 0.1 = 0.03348$$

3.3% the system is being idle.

- **Probability of the system is busy**

$$p = (1 - p0) = 0.967$$

96% the system is busy.

- **Expected number of requests in the categorizer**

$$Lq(q1) = Ls(q1) - P(q1) = 0.05 \text{ requests.}$$

- **Expected waiting time spend in the categorizer**

$$Wq(q1) = Lq(q1) / \lambda1 = 0.05 \text{ seconds}$$

- **Expected number of requests in the determiner**
 $L_q(q_2) = L_s(q_2) - P(q_2) = 1.13$ requests.
- **Expected waiting time spend in the categorizer**
 $W_q(q_1) = L_q(q_1) / \lambda_2 = 0.57$ seconds
- **Expected number of requests in the visualizer**
 $L_q(q_3) = L_s(q_3) - P(q_3) = 8.1$
- **Expected waiting time spend in the categorizer**
 $W_q(q_1) = L_q(q_1) / \lambda_3 = 8.1$ seconds
- **The probability that are at least two requests in the system**
 $p(n \geq 2) = 1 - p_0 - p_1 = 0.913$
91% there are two requests in the system
- **The probability that there are five requests in the system**
 $p(5) = P^5 p_0 = 0.361$
36% five requests exist in the system.

4.3 Simulating the System using Anylogic

Anylogic is simulation software; it provides an easy way to predict the behavior of a system. It has an easy way to display and simulate an event-based system (anylogic, 2016), the same as our proposed system. We simulated our system using Anylogic, with enterprise tools to present event processes in our system, and to implement our queuing model in a trusted simulation tool. Also, we used analysis tools to display statistics and the expectation of our proposed system. Furthermore, we used java classes to keep a track of events occurred, while

simulating our system. The code snippet below shows how we used java classes in anylogic, to keep a track of a total waiting time in the queues, before displaying the results.

```
double startWaiting;  
double enteredSystem;  
double startWaitingDet;  
double startWaitingVis;  
double currentTime;  
  
.....  
entity.currentTime = time()-entity.enteredSystem;  
.....repeated for each queue  
timeInSystemDistr.add(time()-entity.enteredSystem);
```

Figure (4.2), shows the implementation, and results of simulating our system in Anylogic software.

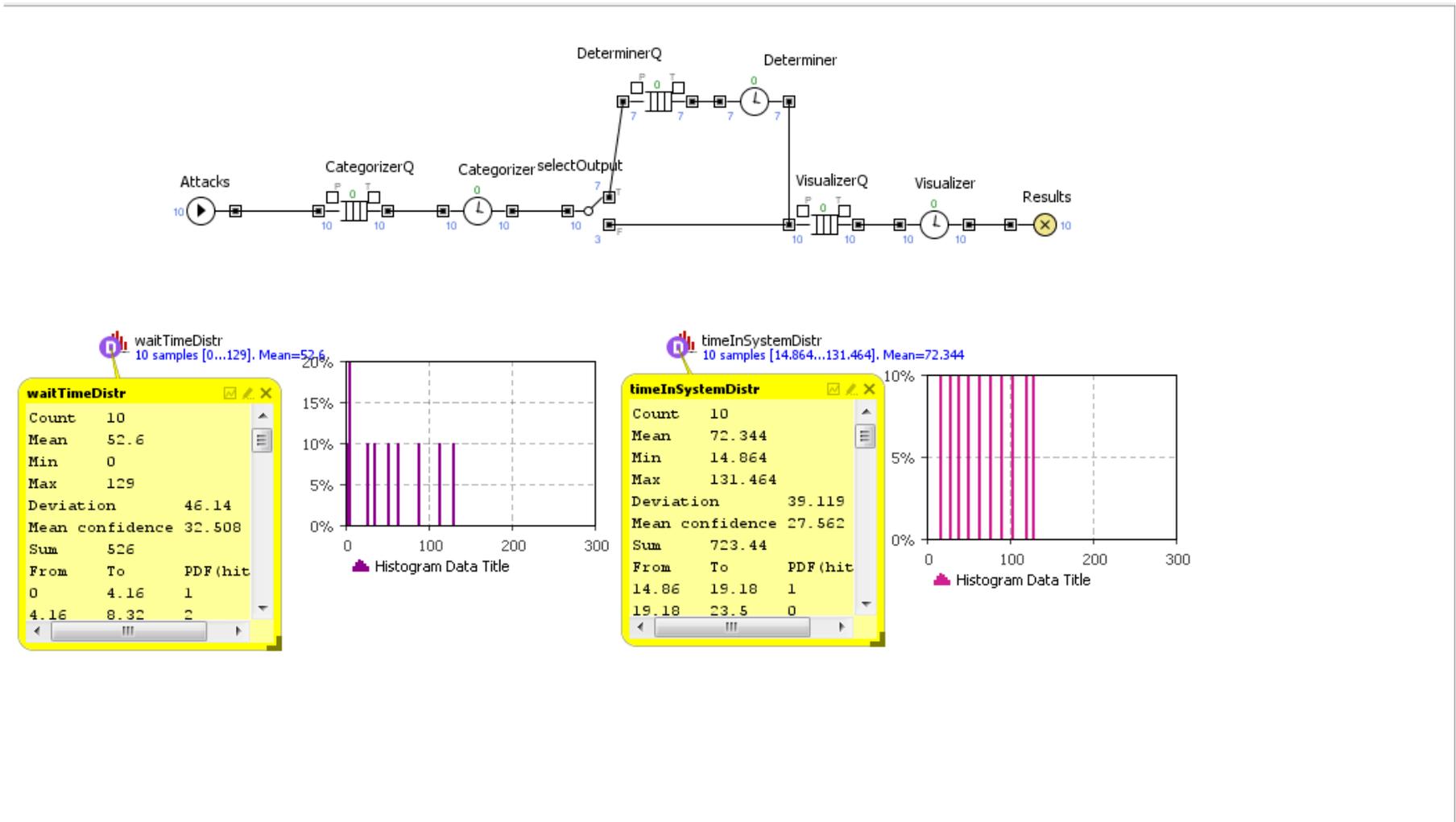


Figure 4.2: Simulating the system using Anylogic.

4.4 System User Interface

The user interface that is displayed to the network administrator consists of; PHP files that are connected to Google visualization libraries, to display information easily and nicely. The main user pages are the annotation bar page, the service display page, the dashboard page and the system update or upgrade page.

4.4.1. Annotation Bar Page

This page display the information in a timeline, to allow the administrator to choose the date, for which attacks occurred in a specific system with all services enabled on it. Figure (4.3) shows the annotation bar page.

Historical Search

Please pick the server and the date to start search :-

Server Name : Date :

Server Desc	Server Services	Danger Status
 s1 (s1)	fingerd(80) macromedia:coldfusion?6(80) trend_micro:interscan_viruswall?3(80) web_page_counter?2(80) snmp(161)	CVSSHIGH



Display History After (1-1-1999)



Show Related Systems

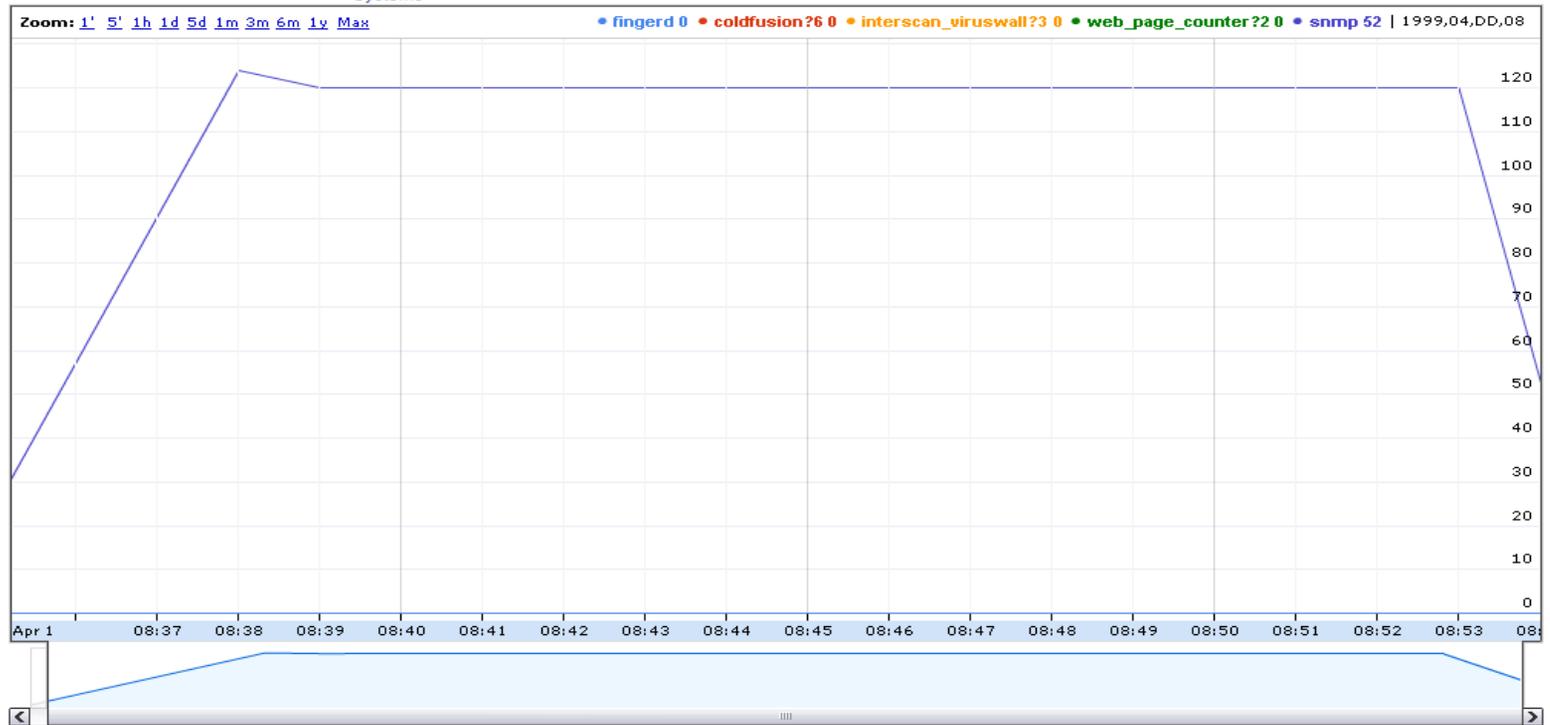


Figure 4.3: Annotation bar page.

4.4.2. Service Display Page

In this page the information is displayed according to a specific service, for the entire DMZ in specific date time interval. It displays information in a bar chart format for a group of events which occurred in slots of dates. Figure (4.4) below demonstrates how information is displayed for a specific service.

Service Search

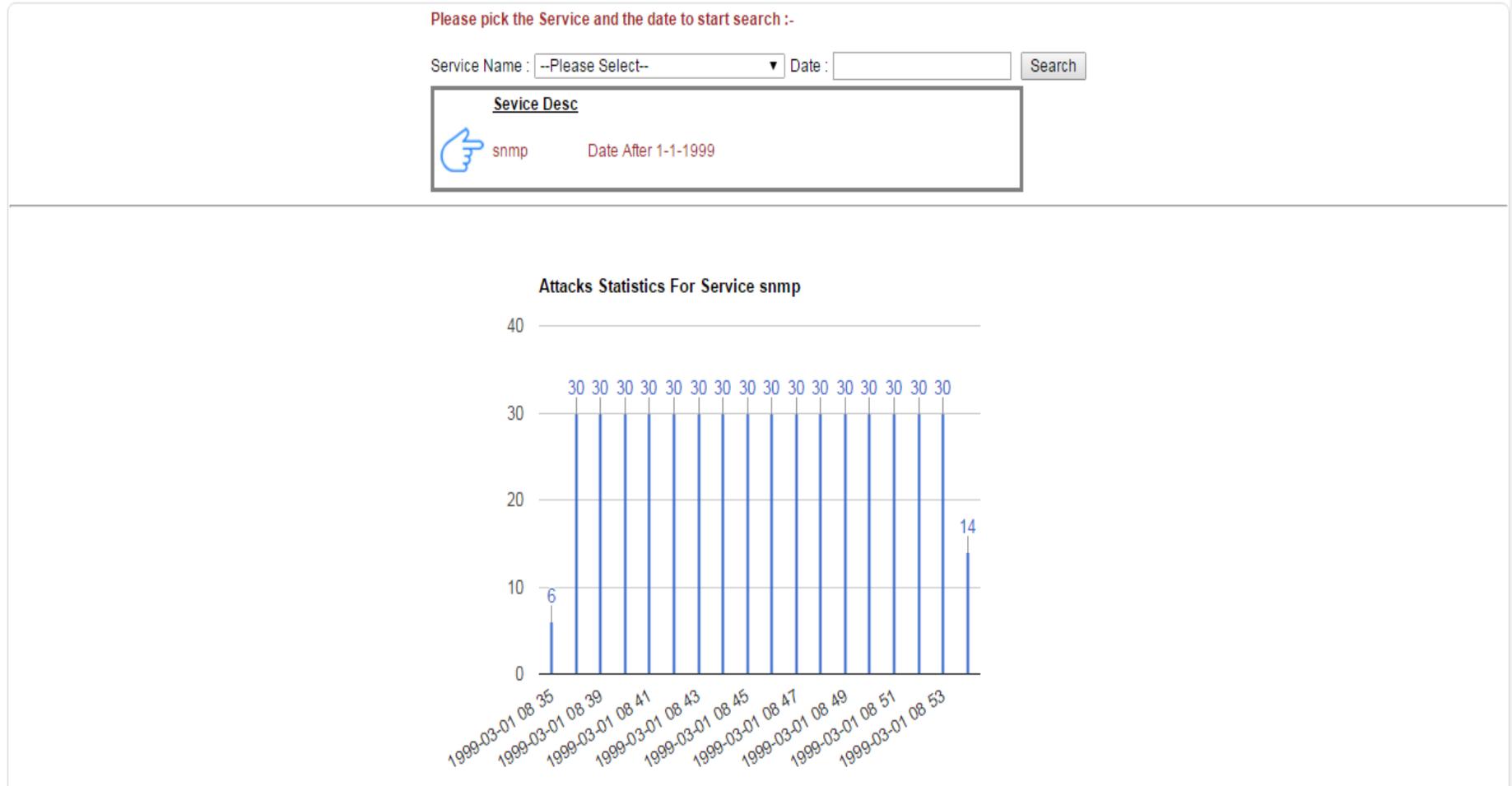


Figure 4.4: Service display page.

4.4.3. Dashboard Page

This main page displays three kind of information. The first type of information is about the dangerous attacks that hit the system services. The second type is to display prediction statistics, about the attacks that may make our system vulnerable. The final type of information is to display the severity and the risk of each attack that hit system services. The first two types of information are displayed in pie chart format, and the later one displayed in a percentage bar chart. Figure (4.5) shows these charts.

Dashboard

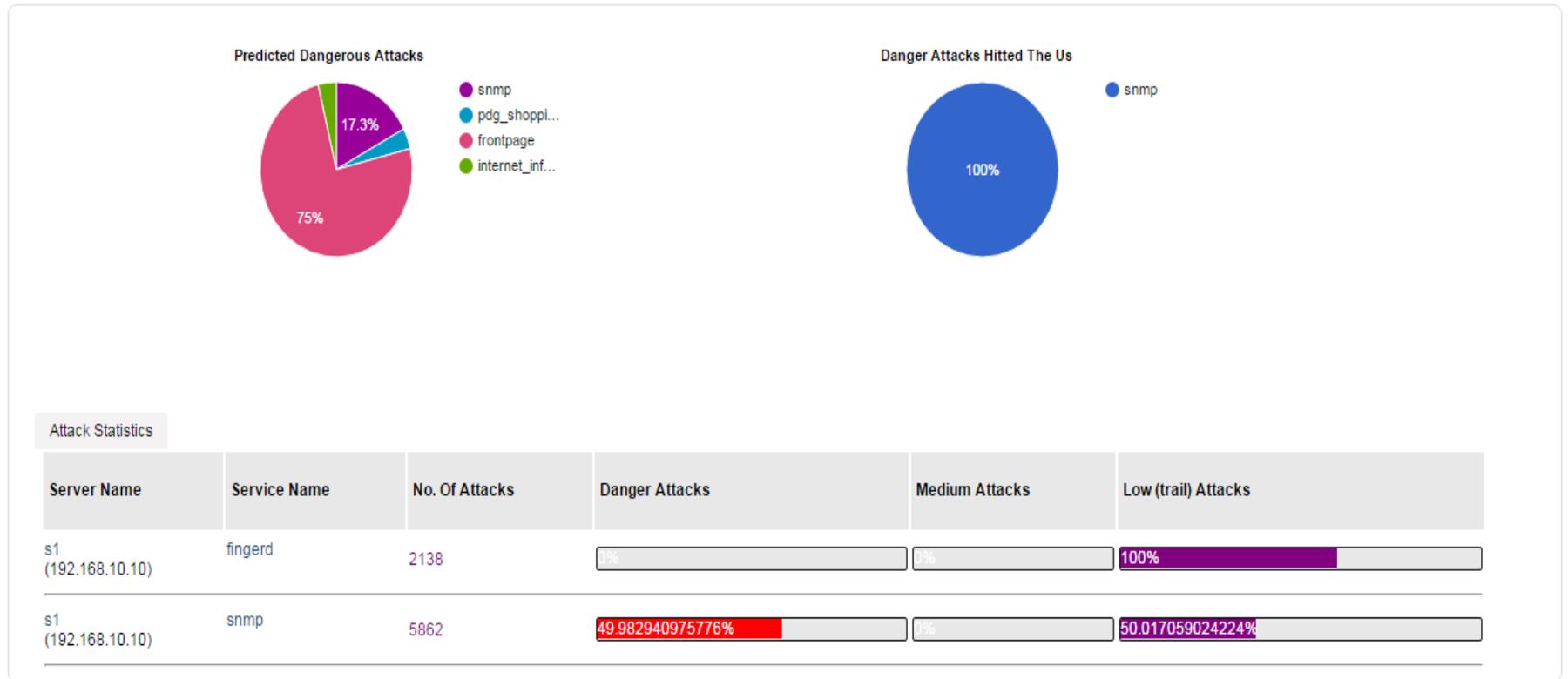


Figure 4.5: Dashboard page.

4.4.4. System Updates / Upgrade page

The system update / upgrade page, allows the administrator to update the vulnerabilities database signature, by adding the latest vulnerability database. It also allows the administrator to update the structure of the system to better suit the organization's needs. Figures (4.6) below demonstrate the mechanism of how the system can be updated or upgraded.

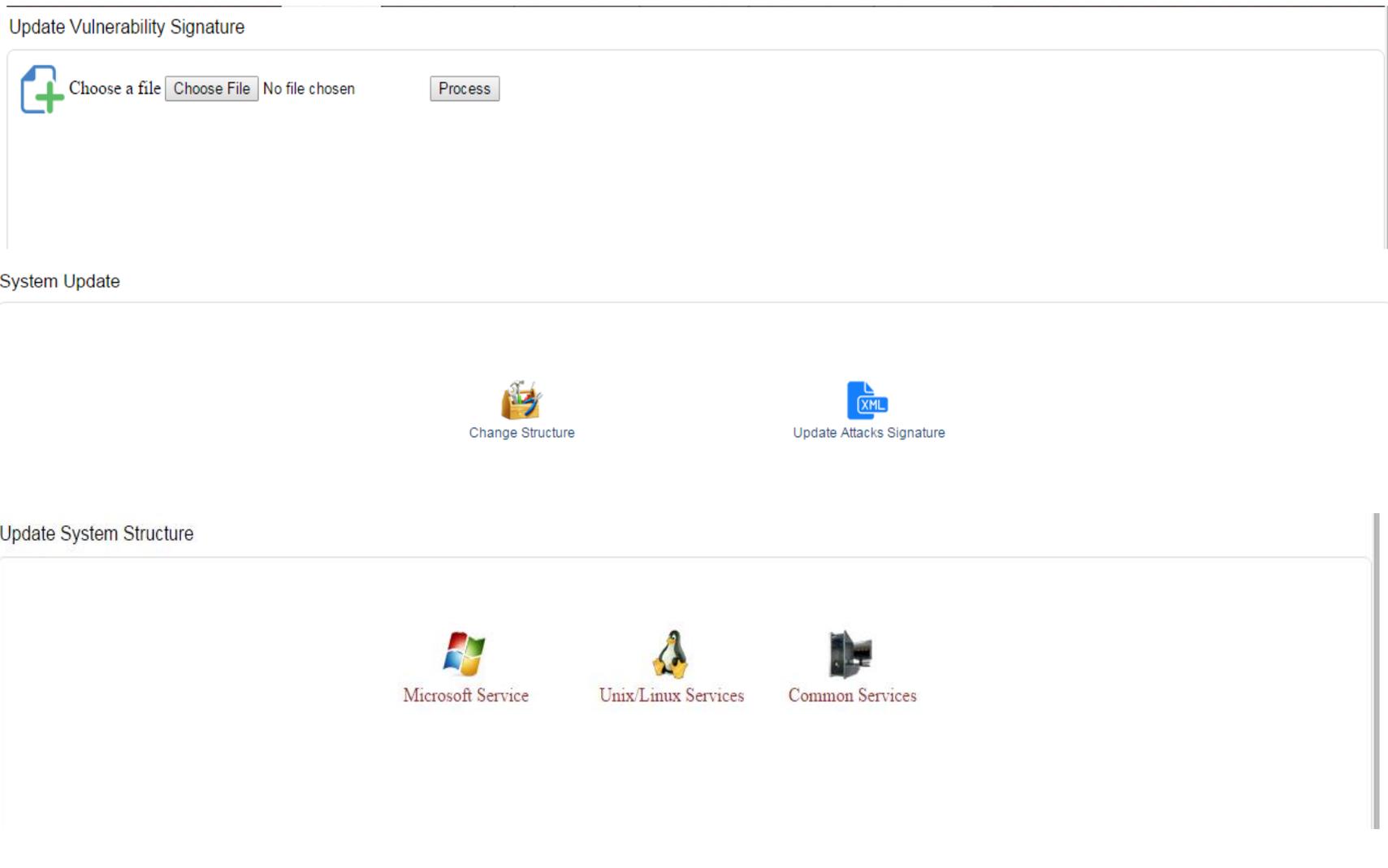


Figure 4.6: System updates / upgrades process.

4.5 Summary

This chapter examined the proposed system by some experiments, and discussed the results for these experiments. We had validated our system in three different ways; KDDCup99, queuing model, and Anylogic simulation.

The results of validating the system using KDDCup99 show that if we increase the threshold of the reasoners the attack prediction increased. Furthermore, the results show that if the number of attacks increased [queried], the processing time increased.

Measuring the performance using queuing model, shows that 96% of the time the system will be busy.

Simulating the system in Anylogic enterprise tools, shows same results as showed in queuing model.

In this chapter we continuously presented some of important page designs that are used in the system. In next chapter, we will conclude, summarize, and suggest some of the future work for the system.

Chapter 5

Summary and Future Work

This chapter concludes the thesis. A review of the importance of the system is presented with focus on the main contributions, results, limitations and assumptions, and possible future work.

In our thesis we modeled a new analysis engine in intrusion detection system for DMZs. DMZs are known as the most important part in the network for each enterprise organization. Our system depends on OKB; which was built based on the basics of the network security attributes [chapter 3], to perform predictions about attacks. We automated the process of extracting useful attacks information, in addition to updating the knowledge of our smart engine by automating the processes of updating and upgrading the system. We believe that adding OKB to the intrusion detection system adds a certain values in terms of making it smarter. This also adds valuable characteristics to the system, in terms of making NIDS perform predictions, which are based on multiple attributes. Thus, the predictions are more accurate [chapter 4].

The rest of this chapter presents the conclusion, including contributions, summary of results, limitations and assumptions. It also presents potential research areas for future work.

5.1 Contribution

In our research, we have two main contributions: extracting useful information that semantically connects snort network intrusion detection system [IDS] logs, and the National Vulnerability Database [NVD] using semantic web technology, so that the signature based IDSs become almost anomaly based IDSs because of the prediction ability. The other contribution is to automate the way of updating ontology structure that is specialized for computer network services. These contributions are summarized in the following subsections.

5.1.1. Extracting Useful Information from Snort NIDS and NVD

In our proposed system we have two main sources of information; snort logs, which are sniffed from the network traffic then logged, and the NVD vulnerabilities databases that are converted into Ontology Knowledge Base [OKB]. In the smart analysis engine for DMZ, we read the attacks which hit our systems from snort logs, after that reasoning and querying these attacks from the stored OKB file, to get extra useful information about each attack recorded.

5.1.2. Automatic Ontology Updates

OKB in analytic engine represent the heart of it because from it we can predict and extract additional useful information about vulnerabilities attacks. OKB consists of two parts; the schema part [T-Box], and the data part [A-Box]. Ontology in our system represents the schema and the relations between vulnerabilities and services. Ontology should be created carefully, since it reflects the structure where the data should be placed. Ontology creation in most cases is performed by specialized tools; such as protégé. The skeleton of our ontology is created by protégé; it reflects the most common network services existing. Due to the continuously changes in the information technology services, and the needs for organizations differ from each other's. Ontology should reflect the organization's needs to allow better prediction and extraction of information, so that it should be updated each time the organization needs changed. Manual ontology updating is not a practical procedure, and makes the system hard to use, as it requires the user to be a semantic web programmer. In our proposed system, we made the ontology creation automated in an easy way [chapter 4], and that allows the data to be easily added to it and reflects the structure modified.

5.2 Results

Our work is evaluated in three different ways; we apply KDDCup99 dataset to our system, and we model our system using queuing model then simulate it using Anylogic simulator. The following subsections summarize how our system is evaluated.

5.2.1. Measures using KDD Benchmark

We used and queried data existing in this dataset, to predict new and related attacks from the created OKB. We performed different tests on these data to get prediction. The results are found as follows:

- The accuracy of prediction depends on several factors that are queried and reasoned [chapter 4].
- As the number of reasoner threshold increased the prediction accuracy increased.
- The best number for the reasoner threshold is four, which shows low false alarm predictions.
- The prediction accuracy for the system is 92.8%, if we set the reasoner threshold to four.

5.2.2. Queuing Model

We applied this mathematical model to our proposed analytical engine, to predict waiting time, and how much time each request is taken to be processed by different modules of the system. This is useful model for a multiuser system, and heavy network systems. The following results showed after applying queuing model to the system:

- Total throughput = 2.9 requests.
- Expected number of requests in the system = 10.8 requests.
- Waiting time for the next request to be served = 11.04 Sec.
- 96% of the time, the system will be busy.

Also we simulated our system using Anylogic software, and showed almost the same results [chapter 4].

5.3 Limitations and Assumptions

All versions of KDDCup99 datasets are combined and used for evaluating our system. These datasets share common attacks signatures. In addition, we used NVD vulnerabilities databases from 2002 to 2008. It is impossible to test all kind of attacks existing in the world. But we think that all attacks can be queried, and informed in the same way as our proposed system performs it, since each attack in the world has behaviors and attributes. One limitation in our system is that we depend on snort IDS, in capturing attacks, and some research shows that snort for heavy networks can drop packets (Bulajoul , James, & Pannu, 2013). Another limitation is that snort is a signature based IDS and some attacks; especially web attacks may be not recognized and logged.

5.4 Future Work

Several open issues can be worked on, such as work on enhancing the performance of data processing; especially by the categorizer module in the system. In addition, work could be undertaken to enhance the predictions performed by the determiner. Also, anomaly based intrusion detection systems can be used, to perform detection on masqueraded attacks.

Furthermore, further research could enable attacks confirmation, by enabling automatic penetration test for each attack that has high risk severity. Other work could integrate the system with firewalls; for example, if attack that is logged and reasoned with high severity, will notify the firewall to automatically create a rule to block the traffic comes from that destination.

References

1. Elahi, G., Yu, E., & Zannone, N. (2009). A Modeling Ontology for Integrating Vulnerabilities into Security Requirements Conceptual Foundations. *Springer Berlin Heidelberg*, 99-114.
2. *Ontology*. (2012). Retrieved 2016, from Semantic Web:
<http://semanticweb.org/wiki/Ontology>
3. anylogic. (2016). *about-us*. Retrieved March 27, 2016, from anylogic:
<http://www.anylogic.com/about-us>
4. Apache Jena. (2015). *Apache Jena Tutorial*. (Apache Jena Co.) Retrieved February 20, 2016, from Apache Jena: https://jena.apache.org/getting_started/index.html
5. Bairwa, S., Mewara, B., & Gajrani, J. (2014). VULNERABILITY SCANNERS: A PROACTIVE APPROACH TO ASSESS WEB APPLICATION SECURITY. *SCNDS 2013 Conference* (pp. 113-124). Ajmer: International Journal on Computational Sciences & Applications (IJCSA).
6. Belch, G., Greiner, S., de Meer, H., & Trivedi, K. S. (1998). Chapter 13. Applications. In G. Belch, S. Greiner, H. de Meer, & K. S. Trivedi, *Queueing Networks and Markov Chains* (pp. 603-678). New York: John Wiley & Sons, Inc.
7. Belch, G., Greiner, S., de Meer, H., & Trivedi, K. S. (1998). *Queueing Networks and Markov Chains*. New York: John Wiley & Sons, Inc.
8. Bulajoul, W., James, A., & Pannu, M. (2013). Network Intrusion Detection Systems in High-Speed Traffic in Computer Networks. *e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on* (pp. 168-175). Coventry: IEEE.
9. C. C. Palmer; IBM Research Divisio. (2001). Ethical hacking. *IBM Systems Journal*, 769-780.

10. Cornell University. (n.d.). *Legal Information Institute*. Retrieved January 2, 2016, from Cornell University Law School: <https://www.law.cornell.edu/uscode/text/44/3542>
11. D. Kshirsagar, D., & Kumar, S. (2013). Ontology for Detection of Web Attacks. *Communication Systems and Network Technologies (CSNT), 2013 International Conference on* (pp. 612-615). Gwalior: IEEE.
12. Ec-Council. (2010). *Ethical Hacking and Countermeasures*. New Mexico: Ec-Council Co.
13. Google org. (2015). *Charts*. (Google) Retrieved March 6, 2016, from Google: <https://developers.google.com/chart/>
14. Handong Wu, J. F. (2009). NETWORK INTRUSION DETECTION AND ANALYSIS SYSTEM AND METHOD.
15. Hoque, M. S., Mukit, M. A., & Bikas, M. A. (2012). AN IMPLEMENTATION OF INTRUSION DETECTION SYSTEM USING GENETIC ALGORITHM. *International Journal of Network Security & Its Applications*, 4(2), 109-120.
16. Horridge, M. (2011). *A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3*. Machester: University of Machester.
17. IC3. (2014). *2014 Internet Crime Report*. IC3.
18. Kotikela, S., Kavi , K., Gomathisankaran, M., & Singhal, A. (2013). VULCAN: Vulnerability Assessment Framework for Cloud Computing. *Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on* (pp. 218-226). Gaithersburg, MD: IEEE.
19. Nist. (2016). *National Vulnerability Database*. Retrieved 2 March, 2016, from National Vulnerability Database: <https://nvd.nist.gov/>

20. nmap CO. (2016). *exploitation tools*. Retrieved March 12, 2016, from SecTools:
<http://sectools.org/tag/splotts/>
21. nmap CO. (2016). *sectools*. Retrieved 2016, from vulnerability scanners:
<http://sectools.org/tag/vuln-scanners/>
22. Panda, M., & Patra, R. M. (2007). NETWORK INTRUSION DETECTION USING NAÏVE BAYES. *IJCSNS International Journal of Computer Science and Network Security*, 7(12), 258-263.
23. Parker, D. (1995). Using threats to demonstrate the elements of information security. *Security and Detection, 1995., European Convention on* (pp. 11-17). Brighton: IET.
24. Phong, C. T., & Yan, W. Q. (2014). An Overview of Penetration Testing. *International Journal of Digital Crime and Forensics*, 50-74.
25. Quinn, S., Waltermire, D., Johnson, C., Scarfone, K., & Banghart, J. (2009, November). The Technical Specification for the Security Content Automation Protocol (SCAP):SCAP Version 1.0. Gaithersburg, USA: NIST.
26. Rehman, R. U. (2003). CHAPTER 3: Working with Snort Rules. In R. U. Rehman, *Intrusion Detection with SNORT: Advanced IDS Techniques Using SNORT, Apache, MySQL, PHP, and ACID* (pp. 75-129). New York, United States Of America: Prentice Hall Professional.
27. Roesch, M., Green, C., Sourcefire Inc, & Cisco. (2015). *SNORT Users Manual*. (amazonaws) Retrieved December 1, 2015, from manual-snort-org: <http://manual-snort-org.s3-website-us-east-1.amazonaws.com/>
28. Saad, S., & Traore, I. (2010). Method ontology for intelligent network forensics analysis. *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on* (pp. 7-14). Ottawa, ON: IEEE.

29. Salini, & Shenbagam. (2015). Prediction and Classification of Web Application Attacks using Vulnerability Ontology. *International Journal of Computer Applications*, 116, 42-47.
30. Semantic Web. (2012, June 13). *Ontology*. Retrieved February 1, 2016, from Semantic Web: <http://semanticweb.org/wiki/Ontology>
31. Stanford. (2015, November 3). *Protege Desktop*. Retrieved March 21, 2016, from protegewiki.stanford: <http://protegewiki.stanford.edu/wiki/Protege4UserDocs>
32. Stanford. (2015, November 10). *WebProtégé*. (stanford) Retrieved March 21, 2016, from rotegewiki.stanford: <http://protegewiki.stanford.edu/wiki/WebProtegeUsersGuide>
33. Stanford. (2016, April 11). *WebProtégé*. (stanford) Retrieved May 1, 2016, from protegewiki.stanford: <http://protegewiki.stanford.edu/wiki/WebProtege>
34. Taye , M. M. (2010). Understanding Semantic Web and Ontologies: Theory and Applications. *Journal of Computing*, 182-192.
35. Uddin, M., Khowaja, K., & Abdul Rehman, A. (2010). Dynamic Multi-Layer Signature Based Intrusion Detection System Using Mobile Agents. *International Journal of Network Security & Its Applications (IJNSA)*, 2(4), 129-141.
36. W3C org. (2015). *Inference*. Retrieved February 24, 2016, from W3C: <https://www.w3.org/standards/semanticweb/inference>
37. Wang, J. A., & Guo, M. (2009). OVM: An Ontology for Vulnerability Management. *CSIIRW- Cyber Security and Information Intelligence Research* (pp. 34-38). New York: ACM.
38. Wang, J. A., Guo, M., Wang, H., Xia, M., & Zhou, L. (2009). Ontology-based Security Assessment for Software Products. *CSIIRW '09 Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies* (pp. 1-10). New York: ACM.

39. Wikipedia. (2016, January 2). *Information security*. Retrieved January 2, 2016, from Wikipedia: https://en.wikipedia.org/wiki/Information_security
40. Wikipedia. (2016, January 1). *Intrusion prevention system*. Retrieved January 28, 2016, from Wikipedia: https://en.wikipedia.org/wiki/Intrusion_prevention_system
41. Wikipedia. (2016, February 13). *Queueing theory*. Retrieved March 7, 2016, from Wikipedia: https://en.wikipedia.org/wiki/Queueing_theory

Appendix 1

Snort

Snort is an open source cross platform network intrusion detection system. It works in one of the following three modes: the sniffer mode, which is simply, sniffs the network traffic, and displays the sniffed traffic on the screen. The second mode is Packet mode, which simply sniffs the traffic, and saves them to the hard disk. The final mode is Network Intrusion Detection Mode [NIDS], which inspects and logs traffic in the network (Uddin, Khowaja, & Abdul Rehman, 2010) (Roesch, Green, Sourcefire Inc, & Cisco, 2015).

Snort uses rules for detection of malware in network traffic. These rules can be applied to snort, and customized to detect new attacks signatures. In addition, we can write a rule to pass, or drop, or log a certain packets based on their payloads. Rules simply understandable syntax; a simple line, and can be extended to multiple lines. In all cases, these rules must be included in the snort configuration file called snort.conf file. Snort rules, have two parts: the rule header, and rule options. The rule header contains actions that should be taken about a specific packet. It is logically divided into Action, Protocol, Address, Port, Direction, Address, and port. The rule options, should be true, to invoke an action in rule header part (Roesch, Green, Sourcefire Inc, & Cisco, 2015) (Rehman, 2003).

The following example is a simple snort rule, which logs all UDP traffic except for source port number 161, which is a SNMP protocol.

```
log udp any !23 -> any any log udp
```

Appendix 2

Protégé

Protégé is an open source cross platform written in java programming language. This tool can be used to create an Ontology Knowledge Base [OKB]. Furthermore, it can be used to query information about stored OKB. Protégé is a GUI tool, which either works through a web browser, or through a desktop application.

Web protégé, supports OWL 2.0, allows collaboration with different users. It also, allows a customizable user interface, and supports multiple ontologies format, such as XML/RDF, Turtle, WL/XML, OBO, and others.

Desktop protégé has the same features as web protégé. In addition, it has support of reasoners. Furthermore, it has a built in capabilities that allow the developer to extend the functionality of protégé, by updating its core (Stanford, 2015) (Stanford, 2016) (Stanford, 2015) (Horridge, 2011). The interface of protégé is easy to use, and consists of: menu, tabs, work area, and selection area .The following example shows the main interface of ontGraph in protégé desktop tool.

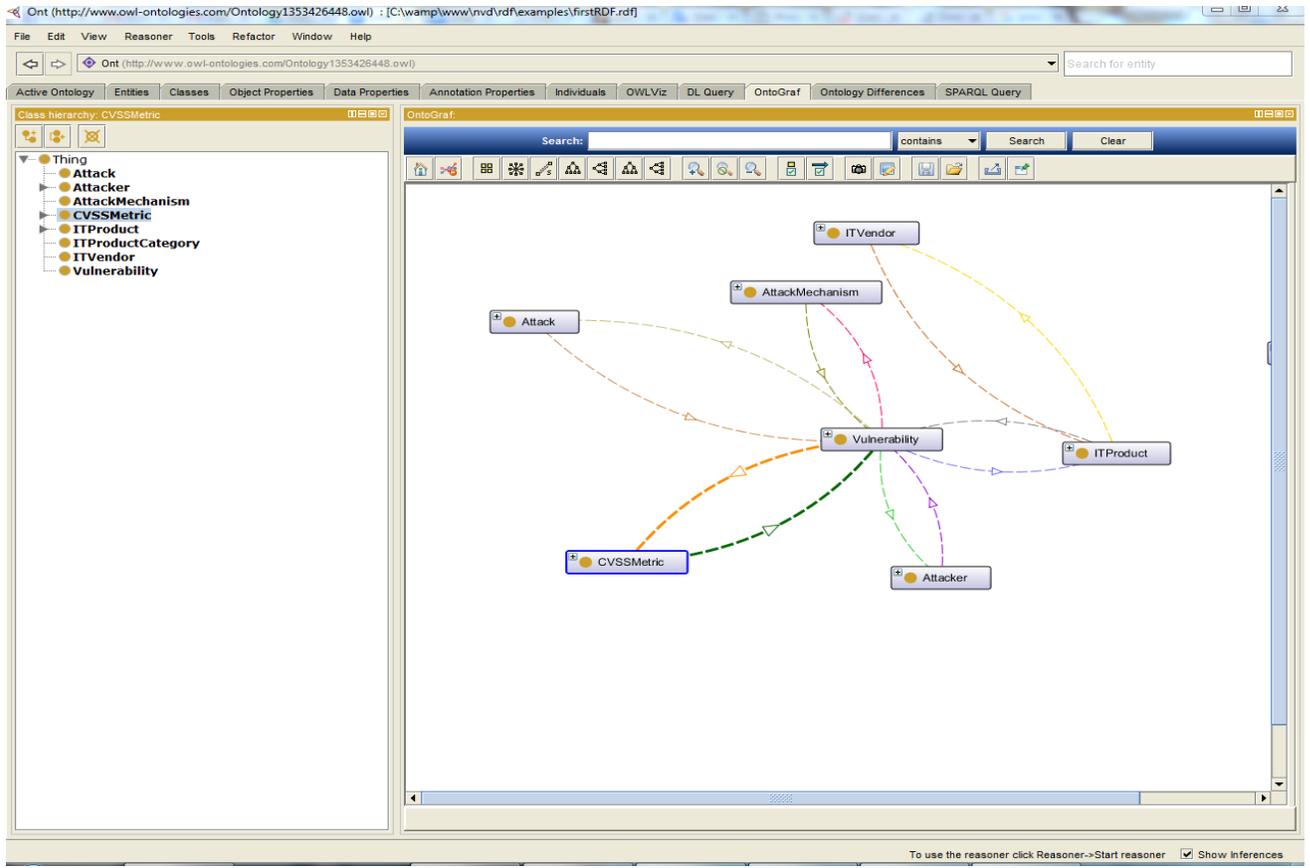


Figure A.1.1: protégé Ontgraph area.

النظام الذكي المختص بأنظمة مراقبة التسلل الإلكترونية بمناطق خوادم الشبكة

إعداد: " محمد رامي" زهير حسن سعيقان

إشراف: د. جهاد النجار و د. رشيد جيوسي

ملخص:

التنبأ بالهجمات الإلكترونية الممكن حصولها على الشبكات الإلكترونية هو مهمة صعبة لأنظمة الحماية الحالية، حيث أن أنظمة المراقبة الحالية تواجه مشكلة في التنبأ بما يمكن الحصول في الشبكة من تهديدات واطار، وهناك العديد من الابحاث التي يتم اجرائها على هذه الانظمة لجعلها اكثر قدرة على التنبأ من خلال خوارزميات تعليم الآلة "Machine Learning".

من خلال البحث الحالي تم إستحداث طريقة جديدة لجعل هذه الانظمة تستبط المعلومات وتحليلها من خلال دمج أنظمة مراقبة التسلل الإلكترونية مع التقنيات الدلالية "Semantic Technology". حيث قمنا بربط أنظمة مراقبة التسلل الإلكترونية دلاليا مع قواعد بيانات الإختراق العالمية " National Vulnerability Database", لجعل هذه الأنظمة دلاليا تستبط معلومات الأخطار المتوقع حصولها على الشبكة الإلكترونية ومعرفة الأنظمة المعرضة لهذة الأخطار. وقمنا ببناء النظام التحليلي للنظام من خلال تطبيق معايير أمن شبكات الحاسوب الإلكترونية وإضافة معايير أخرى مختصة بأنظمة الحاسوب الموجود فيها خوادم المؤسسات, كما وقمنا بجعل هذا النظام قابل للتعديل والتحديث بناء على متطلبات المؤسسة.

وفي مرحلة التقييم قمنا بتطبيق النظام بإستخدام أربعة طرق: بناء نظام ويب و قمنا بتجربة هذا النظام على مجموعة بيانات عالمية تسمى "KDDC up 99 data set" لتوضيح دقة النتائج التي يقوم بها النظام في التنبأ على هذه البيانات والتخفيف من النتائج الغير صحيحة الناتجة من التنبأ، كما وقمنا بمحاكاة النظام باستخدام تقنية الطوبير "Queuing Model" لمعرفة قدرة النظام على العمل في البيئة الإلكترونية الضخمة, وقمنا بإختبار نظام الطوابير باستخدام اداة محاكاة مختصة تسمى "Anylogic".