SEMANTIC MAPPING OF SECURITY EVENTS TO KNOWN ATTACK PATTERNS

Xiao Ma

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

August 2018

Abstract

Semantic Mapping of Security Events to Known Attack Patterns

Xiao Ma

In order to provide cyber environment security, analysts need to analyze a large number of security events on a daily basis and take proper actions to alert their clients of potential threats. The increasing cyber traffic drives a need for a system to assist security analysts to relate security events to known attack patterns. This thesis describes the enhancement of an existing Intrusion Detection System (IDS) with the automatic mapping of snort alert messages to known attack patterns. Our system relies on three approaches: supplementing snort messages by adding related Common Vulnerabilities and Exposures (CVE) entities, pre-clustering similar snort messages before mapping them to attack patterns in Common Attack Pattern Enumeration and Classification (CAPEC) and using Latent Semantic Analysis (LSA) to reduce the dimension of the feature space. The module has been deployed in our partner company and when evaluated against the recommendations of two security analysts, it improved the F-measure of their system from 51.81% to 64.84%.

Acknowledgments

First of all, I would like to express my sincere thanks to my supervisor Dr. Leila Kosseim. Her patient guidance and excellent feedback at every step of the research contributed inestimably to my thesis and inspired me to overcome every difficulty. It was a great honor for me to be supervised by my supervisor, whom I profoundly admire her outstanding knowledge. I am also extremely thankful for her consideration, kindness and understanding. I believed I have learned a lot from my supervisor and these two years of research.
Also, I would like to thank Elnaz Davoodi and Nicandro Scarabeo. Their experienced analysis, advice and suggestions greatly improved understanding for the project. Every discussion with them offered me new ideas and encouraged me to move forward. I quite enjoyed our discussions.
I also would like to express my thanks to the examining committee, Professors Brigitte Jaumard and Olga Ormandjieva, as well as the chair Professor Yuhong Yan, for their helpful feedback and insightful comments.
Finally, I would like to thank my fellow lab mates at the Computational Linguistics at Concordia (CLaC). Their support, interesting discussions and valuable contributions enriched my knowledge and boosted my research during these two years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Motivation

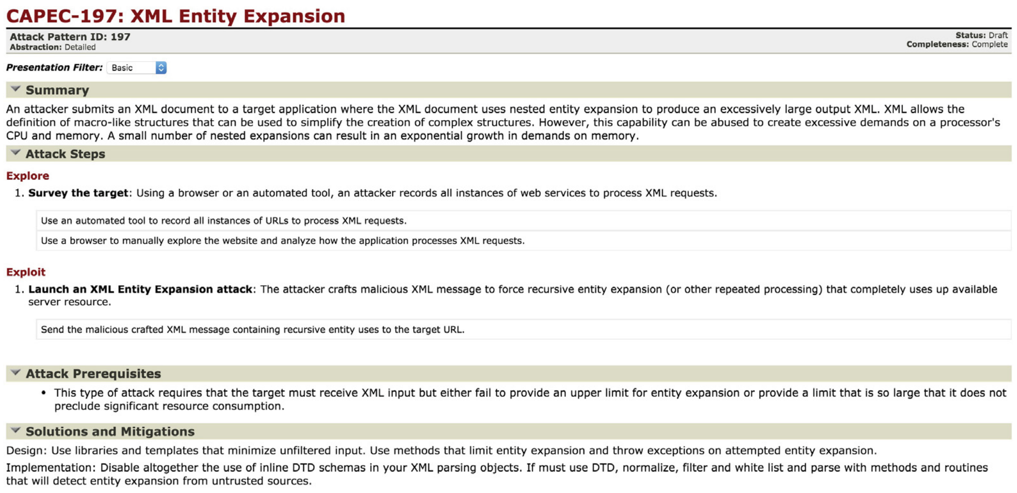
With the increasing dependence on computer infrastructure, cyber security has become an essential service for organizations, governments and individuals. Cyber security refers to the collection of tools, approaches and technologies which are used to prevent unauthorized behaviour, hacking attacks, denial of service, malware propagation and other anomalies [Schatz et al., 2017]. In order to detect and prevent harmful behaviour, multiple sensors are typically installed in network infrastructures. Each sensor is equipped with several security systems, such as Intrusion Detection Systems (IDS)[1] and Assert Detection Tools [Ashoor and Gore, 2011]. These systems perform network traffic analysis in real-time, detect suspicious activities and produce alert messages. These messages, triggered by suspicious activities, are called security events. Snort [Roesch et al., 1999] is a widely used lightweight IDS system installed in many sensors. By capturing and decoding suspicious Transmission Control Protocol/Internet Protocol (TCP/IP) packets, snort generates short messages regarding network traffic data to facilitate the task of security analysts to recognize suspicious behaviours and act accordingly [Roesch et al., 1999]. Figure 1 shows seven examples of snort messages.

Once a software vulnerability or weakness is identified, it constitutes very useful information for security analysts. It is therefore inventoried, classified and made publicly available in large repositories such as the U.S. National Vulnerability Database

[1]It is a device or an application which monitors a network or system malicious activities or policy violations, reporting to an administrator.

1. *FILE-OTHER XML exponential entity expansion attack attempt*
2. *FILE-IDENTIFY RealPlayer skin file attachment detected*
3. *SQL generic sql exec injection attempt - GET parameter*
4. *FILE-OTHER XML exponential entity expansion attack attempt*
5. *MALWARE-OTHER Win.Exploit.Hacktool suspicious file download*
6. *BROWSER-PLUGINS MSN Setup BBS 4.71.0.10 ActiveX object access*
7. *BROWSER-IE Microsoft Internet Explorer asynchronous code execution attempt*

Figure 1: Examples of 7 snort messages



CAPEC-197: XML Entity Expansion

Attack Pattern ID: 197
Abstraction: Detailed

Status: Draft
Completeness: Complete

Presentation Filter: Basic

Summary

An attacker submits an XML document to a target application where the XML document uses nested entity expansion to produce an excessively large output XML. XML allows the definition of macro-like structures that can be used to simplify the creation of complex structures. However, this capability can be abused to create excessive demands on a processor's CPU and memory. A small number of nested expansions can result in an exponential growth in demands on memory.

Attack Steps

Explore

1. **Survey the target**: Using a browser or an automated tool, an attacker records all instances of web services to process XML requests.

Use an automated tool to record all instances of URLs to process XML requests.
Use a browser to manually explore the website and analyze how the application processes XML requests.

Exploit

1. **Launch an XML Entity Expansion attack**: The attacker crafts malicious XML message to force recursive entity expansion (or other repeated processing) that completely uses up available server resource.

Send the malicious crafted XML message containing recursive entity uses to the target URL.

Attack Prerequisites

- This type of attack requires that the target must receive XML input but either fail to provide an upper limit for entity expansion or provide a limit that is so large that it does not preclude significant resource consumption.

Solutions and Mitigations

Design: Use libraries and templates that minimize unfiltered input. Use methods that limit entity expansion and throw exceptions on attempted entity expansion.

Implementation: Disable altogether the use of inline DTD schemas in your XML parsing objects. If must use DTD, normalize, filter and white list and parse with methods and routines that will detect entity expansion from untrusted sources.

Figure 2: Example of a CAPEC Attack Pattern

(NVD) [Division, 2017], Common Vulnerabilities and Exposures (CVE) [MITRE, 2017b] and Common Attack Pattern Enumeration and Classification (CAPEC) [MITRE, 2017a]. For example, Figure 2 shows an attack pattern in CAPEC. This pattern shows the typical scenario of a network attack via an *XML Entity Expansion*. The pattern is composed of several fields that describe the vulnerability in natural language. These software vulnerability repositories constitute vital information for security analysts as they can be used to verity if the snort messages describing current network activities seem to correspond to known attack patterns.

Recognizing suspicious behaviours from snort messages is a difficult task as the messages are short and contain very little natural language. For example, the first snort message in Figure 1 only contains 7 tokens. Thus, today this task is still mostly performed by human security experts. After security analysts from our industrial partner[2] analyzed the first snort message, they associated it to the CAPEC attack pattern of Figure 2. This allowed them to extract the corresponding solutions specified in the `Solutions and Mitigations` field in the attack pattern and communicate it to their clients.

Because of the increasing volume of network traffic, the workload of security analysts has become much heavier and the possibility of not detecting a security risk has become critical. In order to allow security analysts to better assess risks, the automatic mapping of security events to known attack patterns is desired. **The goal of this thesis is to propose an automatic method to map snort messages, as shown in Figure 1, to attack patterns as shown in Figure 2.**

1.2 Problem Statement

This thesis is the result of an industrial collaboration with the company Above Security. The company uses a proprietary *Security Information and Event Management (SIEM)* system to assist security analysts. Figure 3 illustrates a typical SIEM system workflow. As [Scarabeo et al., 2015] describes, the SIEM captures security events in the so-called *collection layer*, and uses pre-defined rules stored in a knowledge base through the *intelligence layer*. Each security event is matched by rules in the knowledge base in order to identify sensitive events. Then, security analysts observe the matching events in the *analysis layer* and respond to clients by writing reports in the *response layer*. With the increasing volume of security events, the daily updates of the pre-defined rules in the knowledge base has become necessary. However, this task is usually done in an ad-hoc fashion as there is no standard basis for the definition of these rules. Thus, a system which can recommend attack patterns to security analysts is welcome. The goal of this thesis is to assist security analysts by suggesting the most appropriate attack patterns corresponding to snort messages using the semantic similarity between these two natural language descriptions. After an analysis of our partner's prototype system, the following two research problems have been identified:

1. Evaluate the quality of their prototype system (see Chapter 3)

2. Improve the quality of their prototype system, in particular to account for the small length of snort messages (see Chapters 4, 5 and 6).

[2]This thesis is the result of a MITACS Accelerate project with the company Above Security.

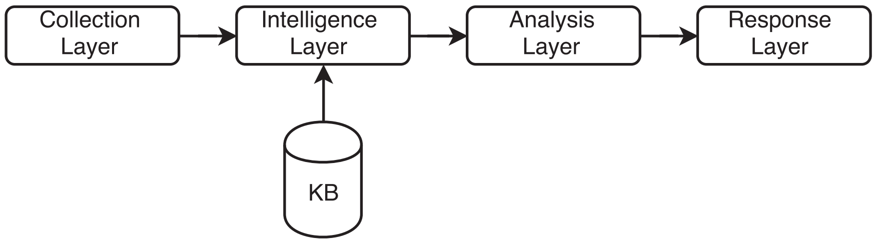


Figure 3: Workflow of a typical SIEM system

Currently, the average size of security events is quite small so that not much information can be extracted from them to be used in building automated model for mapping attacks; whereas attack pattern descriptions are much longer. For instance, the first snort alert message in Figure 1 only has 7 tokens but the size of the corresponding CAPEC entry is 268 tokens (see Figure 2). The difference in length between snort messages and attack patterns significantly reduces the effectiveness of standard semantic similarity measures.

1.3 Contribution

This thesis makes four major contributions:

1. We propose to use standard evaluation metrics to evaluate the quality of the baseline system and our enhanced system. This is described in Chapter 3 and in [Ma et al., 2018].
   In addition, we investigated three approaches to address the issue of the short size of snort messages and improve the performance of our partner's existing mapping system.

2. Supplementing snort messages by adding related Common Vulnerabilities and Exposures (CVE) entities (see Chapter 4).

3. Pre-clustering similar snort messages before mapping them to attack patterns in Common Attack Pattern Enumeration and Classification (CAPEC) (see Chapter 5 [Ma et al., 2018]).

4. Using Latent Semantic Analysis to reduce the dimension of the feature space as well as finding similar semantic descriptions between snort messages and attack patterns in CAPEC (see Chapter 6).

The overall contribution of this thesis is to propose a method to decrease the workload of security analysts via suggesting better quality attack patterns in CAPEC for each security event. We evaluated our enhanced system against a gold-standard created by two security analysts. The original system achieved an F-measure of only 51.81%. Through our enhancements, the F-measure improved to 64.57%. Our partner company was satisfied with the enhanced system and deployed it into production.

1.4 Thesis Outline

In this chapter, we motivated our work and described the problems addressed by our research. The rest of the thesis is structured as follows: Chapter 2 presents an overview of previous work, including the three most used public vulnerability repositories, the ArkAngel system currently used in our partner organization and various techniques to compute sentence similarity in the field of natural language processing. Chapter 3 introduces the evaluation metrics used to evaluate the current system and the methodology we used to evaluate its mapping quality. Based on this evaluation, Chapters 4, 5 and 6 respectively present our experiments to enhance the performance of the system. Finally, Chapter 7 presents conclusions and future work.

Chapter 2

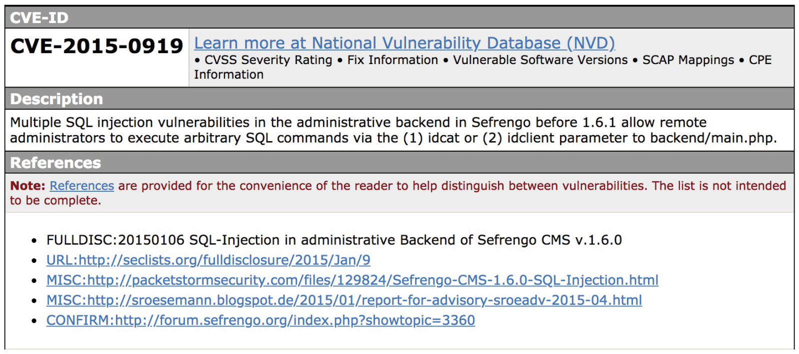
Previous Work

As shown in Figure 3, in a typical Security Information and Event Management (SIEM) system, the *intelligence layer* is responsible for identifying suspicious behaviours. Typically, pre-defined rules and scripts written by experienced security analysts are used to match security events to known risks. However, with the significant increase of network traffic, these pre-defined rules need to be maintained on a daily basis and the workload of security analysts has become much heavier. To address these issues, security analysts in our partner company, Above Security, developed an intelligent system to automatically map security events to attack patterns in the CAPEC repository (see Section 2.2.1).

In this chapter, we present three widely used public vulnerability repositories in Section 2.1; while Section 2.2 describes the workflow of the SIEM system in our partner company. Section 2.3 presents the most common sentences similarity measures used in natural language process (NLP), based on string similarity and semantic similarity.

2.1 Public Repositories of Known Vulnerabilities

Over the years, several public known vulnerability repositories have been developed in the field of cyber security. This section describes three of the most widely used vulnerability repositories: Common Vulnerabilities and Exposures (CVE), National Vulnerability Database (NVD) and Common Attack Pattern Enumeration and Classification (CAPEC).


CVE-ID

CVE-2015-0919 Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

Multiple SQL injection vulnerabilities in the administrative backend in Sefrengo before 1.6.1 allow remote administrators to execute arbitrary SQL commands via the (1) idcat or (2) idclient parameter to backend/main.php.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- FULLDISC:20150106 SQL-Injection in administrative Backend of Sefrengo CMS v.1.6.0
- URL:http://seclists.org/fulldisclosure/2015/Jan/9
- MISC:http://packetstormsecurity.com/files/129824/Sefrengo-CMS-1.6.0-SQL-Injection.html
- MISC:http://sroesemann.blogspot.de/2015/01/report-for-advisory-sroeadv-2015-04.html
- CONFIRM:http://forum.sefrengo.org/index.php?showtopic=3360

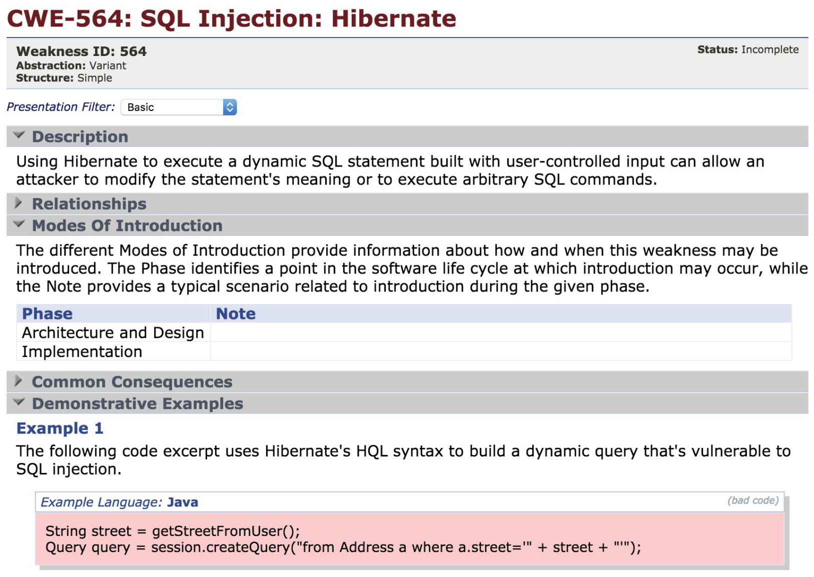
Figure 4: Example of a CVE Entity

2.1.1 Common Vulnerabilities and Exposures

In the 1990s, with the rapid development of computer infrastructure and network facilities, the security of cyber environments became a serious concern. In order to provide protection for their infrastructure, many organizations built their own security vulnerability database products to record known vulnerabilities [MITRE, 2017b]. Each organization used their own regulations and patterns to record software weaknesses so that significant variations existed among these products. This lead to serious issues of cross-product consistency and interoperability. In 1999, in order to address these problems, the Common Vulnerabilities and Exposures (CVE) was launched by the U.S. Department of Homeland Security and to standardize the recording of security vulnerabilities [MITRE, 2017b].

In the CVE, the smallest unit is the *entity*. Each entity records a single known vulnerability and all the entities in the same year are recorded in the order that they were identified. Each entity is composed of a CVE ID, a brief description of the vulnerability and references to their initial announcement. Figure 4 shows an example of a CVE entity representing the *SQL Injection* vulnerability. In this entity, *CVE-2015-0919* is the vulnerability ID, followed by its description and references. Currently, CVE contains a total of 98,375 entities and each entity contains an average of 30 tokens. Because of its wide coverage, most computer security consulting companies use CVE as the known vulnerability repository in order to obtain better vulnerabilities coverage and security protection.

However, the descriptions in CVE entities have often been criticized as being


CWE-564: SQL Injection: Hibernate

Weakness ID: 564
Abstraction: Variant
Structure: Simple

Status: Incomplete

Presentation Filter: Basic

Description

Using Hibernate to execute a dynamic SQL statement built with user-controlled input can allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.

Relationships

Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the software life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
Architecture and Design	
Implementation	

Common Consequences

Demonstrative Examples

Example 1

The following code excerpt uses Hibernate's HQL syntax to build a dynamic query that's vulnerable to SQL injection.

Example Language: **Java** (bad code)

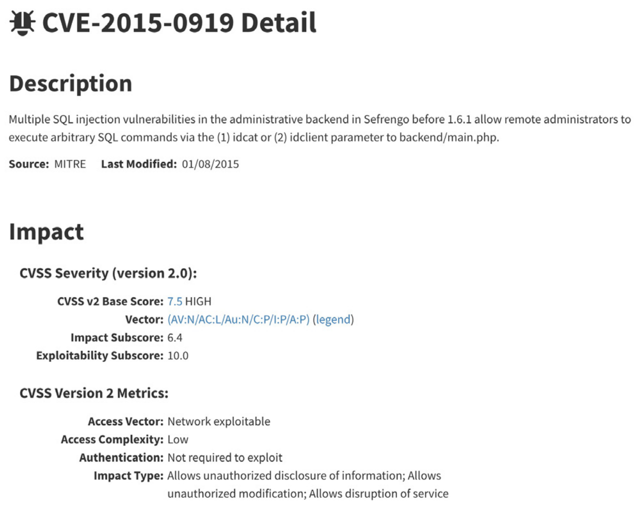
```
String street = getStreetFromUser();
Query query = session.createQuery("from Address a where a.street='" + street + "'");
```

Figure 5: Example of a CWE Entity

too general to guide software developers in writing code without security weaknesses [MITRE, 2017b]. In addition, as the CVE has become quite large (with 98,375 entities), it is not easy to refer to its entities because it does not provide a categorization of known weaknesses. In order to overcome these two issues, a CVE derivation product called Common Weakness Enumeration (CWE) was launched by MITRE's CVE Team in 2005 [MITRE, 2017c]. Each entity in the CWE not only explains the implication of each software weaknesses in terms of software design, usage of software framework and software coding, but it also provides code examples which can cause weaknesses. For example, Figure 5 shows the *SQL Injection* software weakness in the CWE. Compared with the entity in CVE (shown in Figure 4), the CWE entity provides several additional information, such as *Relationships*, *Modes of Introduction*, etc., to describe the *SQL Injection* weakness. Currently, there are 714 CWE entities which are categorized based on a variety of criteria, such as the most frequently searched weaknesses, different programming languages weaknesses, etc.

2.1.2 The National Vulnerability Database

The National Vulnerability Database (NVD) is a U.S. government vulnerability repository managed by the Computer Security Division of the National Institute of Standards and Technology (NIST) [NIST, 2017]. It was launched in 2000 under the initial name of *Internet - Categorization of Attacks Toolkit (ICAT)* as an enhancement of CVE. In the NVD, the smallest unit is called a *feed* and it consists of a CVE entity (see Section 2.1.1), a "technical details" section which links the feed to the CWE entity and an impact metric. The Common Vulnerability Scoring System (CVSS) [FIRST, 2018] is an open framework which measures the security of the known vulnerabilities. For instance, Figures 6 and 7 show the impact metrics and the "technical details" zone for the *SQL Injection* vulnerability (see Figure 4) in an NVD feed. The integration of CWE and CVSS into CVE entries as an NVD feed provides security analysts detailed information for each known vulnerability.



CVE-2015-0919 Detail

Description

Multiple SQL injection vulnerabilities in the administrative backend in Sefrengo before 1.6.1 allow remote administrators to execute arbitrary SQL commands via the (1) idcat or (2) idclient parameter to backend/main.php.

Source: MITRE **Last Modified:** 01/08/2015

Impact

CVSS Severity (version 2.0):

CVSS v2 Base Score:	7.5 HIGH
Vector:	(AV:N/AC:L/Au:N/C:P/I:P/A:P) (legend)
Impact Subscore:	6.4
Exploitability Subscore:	10.0

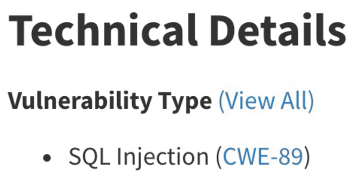
CVSS Version 2 Metrics:

Access Vector:	Network exploitable
Access Complexity:	Low
Authentication:	Not required to exploit
Impact Type:	Allows unauthorized disclosure of information; Allows unauthorized modification; Allows disruption of service

Figure 6: Impact Metrics in an NVD Feed

2.1.3 Common Attack Pattern Enumeration and Classification

In addition to the CVE and CWE repositories, a third vulnerability repository has been released by MITRE, the *Common Attack Pattern Enumeration and Classification (CAPEC)* [MITRE, 2017a]. CAPEC is a more recent and richer repository than


Technical Details

Vulnerability Type (View All)

- SQL Injection (CWE-89)

Figure 7: Technical Details in an NVD Feed

CVE and CWE. While the CWE only provides a list of software weaknesses, CAPEC indicates the common steps that hackers typically use to exploit vulnerabilities, and also provides suggested mitigations for each known software weakness. In CAPEC, the descriptions of known software weaknesses are called *attack patterns*. Currently, CAPEC contains a total of 508 attack patterns represented hierarchically in 9 high level mechanisms of attack categories. A category in CAPEC is a collection of attack patterns in common effects or intents. For example, the *Inject Unexpected Items* category contains the *Code Inclusion* attack pattern, the *Command Injection* pattern, the *Object Injection* pattern, etc. Figure 8 illustrates these 9 categories followed by the category ID in the parenthesis, such as *Collection and Analyze Information*, *Inject Unexpected Items*, *Abuse Existing Functionality*, etc. In each category, there are three kinds of attack patterns:

Meta Attack Pattern
The meta attack pattern is an abstract description of an attack. It describes a known vulnerability at a high level and does not provide detailed techniques used in exploiting the vulnerabilities. For example, the *Command Injection* attack pattern shown in Figure 9 is a meta attack pattern.

Standard Attack Pattern
The standard attack pattern focuses on describing specific methodologies used in an attack. It provides details about a known vulnerability, such as detailed descriptions, attack steps, exploiting techniques, etc. For instance, in Figure 9, the *SQL Injection* and the *XML Injection* are standard attack patterns.

Detailed Attack Pattern
Compared with meta and standard attack patterns, detailed attack patterns provide a more specific descriptions in the attack steps techniques, exploiting

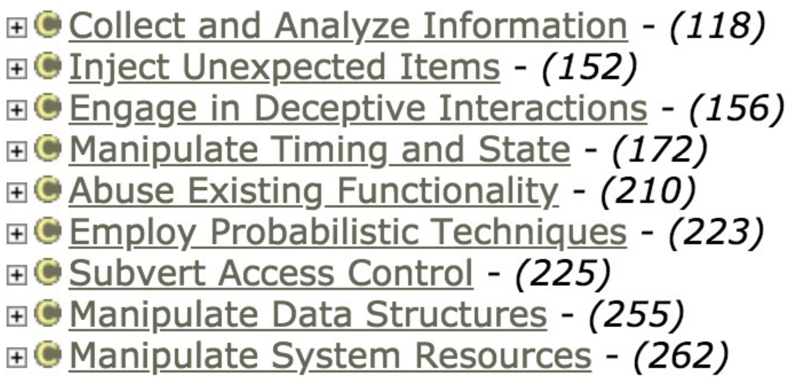

⊞ C Collect and Analyze Information - (118)
⊞ C Inject Unexpected Items - (152)
⊞ C Engage in Deceptive Interactions - (156)
⊞ C Manipulate Timing and State - (172)
⊞ C Abuse Existing Functionality - (210)
⊞ C Employ Probabilistic Techniques - (223)
⊞ C Subvert Access Control - (225)
⊞ C Manipulate Data Structures - (255)
⊞ C Manipulate System Resources - (262)

Figure 8: Categories of Attack Patterns in CAPEC

prerequisites and several code segment samples. In Figure 9 shows, the *Blind SQL Injection* is a detailed attack pattern.

These three types of attack patterns are the basis of the hierarchy representation in CAPEC. For example, Figure 9 illustrates that the *SQL Injection* software weakness is a type of *Command Injection* which itself is a type of the *Inject Unexpected Items* category.

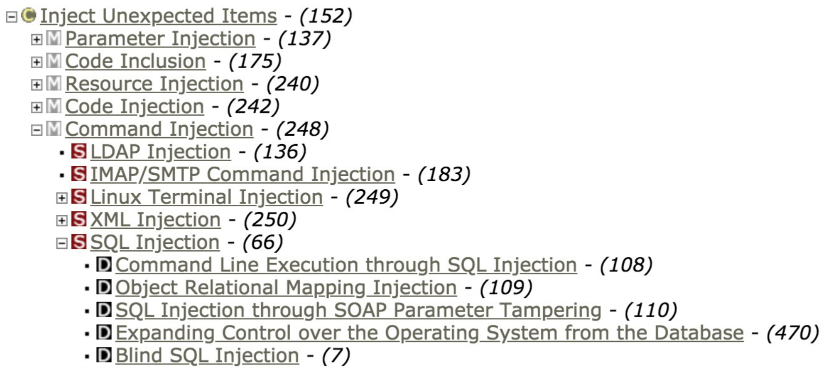
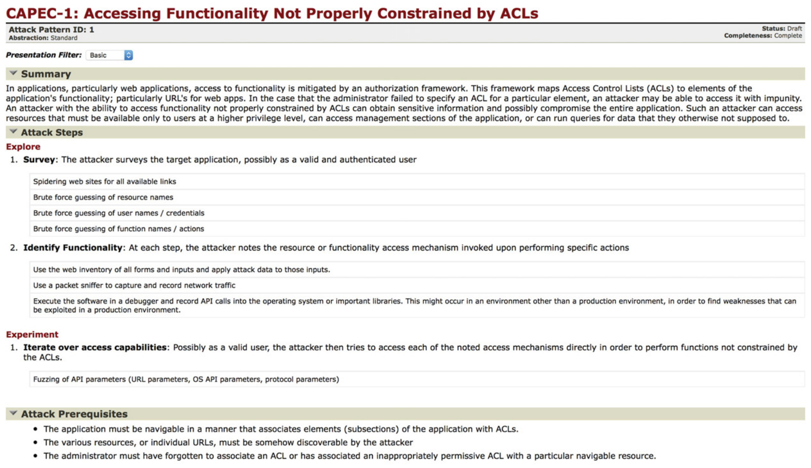

```
⊟ C Inject Unexpected Items - (152)
    ⊞ M Parameter Injection - (137)
    ⊞ M Code Inclusion - (175)
    ⊞ M Resource Injection - (240)
    ⊞ M Code Injection - (242)
    ⊟ M Command Injection - (248)
        • S LDAP Injection - (136)
        • S IMAP/SMTP Command Injection - (183)
        ⊞ S Linux Terminal Injection - (249)
        ⊞ S XML Injection - (250)
        ⊟ S SQL Injection - (66)
            • D Command Line Execution through SQL Injection - (108)
            • D Object Relational Mapping Injection - (109)
            • D SQL Injection through SOAP Parameter Tampering - (110)
            • D Expanding Control over the Operating System from the Database - (470)
            • D Blind SQL Injection - (7)
```

Figure 9: Hierarchy of Attack Patterns in CAPEC

In addition, as shown in Figure 10, an attack pattern is described by several paragraphs. Each paragraph is called a *CAPEC field*. Each pattern is composed of 15 fields that are described in natural language. These includes one *Summary* field, five *Attack_Steps_Survey* fields, four *Attack_Steps_Identify_Functionality* fields, two *Experiments* and three *Attack_Prerequisites* fields. These CAPEC fields can be repeated in different attack patterns. Thus, in total, CAPEC contains 5,096 fields of



CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs

Attack Pattern ID: 1
Abstraction: Standard

Status: Draft
Completeness: Complete

Presentation Filter: Basic

Summary

In applications, particularly web applications, access to functionality is mitigated by an authorization framework. This framework maps Access Control Lists (ACLs) to elements of the application's functionality; particularly URL's for web apps. In the case that the administrator failed to specify an ACL for a particular element, an attacker may be able to access it with impunity. An attacker with the ability to access functionality not properly constrained by ACLs can obtain sensitive information and possibly compromise the entire application. Such an attacker can access resources that must be available only to users at a higher privilege level, can access management sections of the application, or can run queries for data that they otherwise not supposed to.

Attack Steps

Explore

1. **Survey**: The attacker surveys the target application, possibly as a valid and authenticated user

    Spidering web sites for all available links

    Brute force guessing of resource names

    Brute force guessing of user names / credentials

    Brute force guessing of function names / actions

2. **Identify Functionality**: At each step, the attacker notes the resource or functionality access mechanism invoked upon performing specific actions

    Use the web inventory of all forms and inputs and apply attack data to those inputs.

    Use a packet sniffer to capture and record network traffic

    Execute the software in a debugger and record API calls into the operating system or important libraries. This might occur in an environment other than a production environment, in order to find weaknesses that can be exploited in a production environment.

Experiment

1. **Iterate over access capabilities**: Possibly as a valid user, the attacker then tries to access each of the noted access mechanisms directly in order to perform functions not constrained by the ACLs.

    Fuzzing of API parameters (URL parameters, OS API parameters, protocol parameters)

Attack Prerequisites

- The application must be navigable in a manner that associates elements (subsections) of the application with ACLs.
- The various resources, or individual URLs, must be somehow discoverable by the attacker
- The administrator must have forgotten to associate an ACL or has associated an inappropriately permissive ACL with a particular navigable resource.

Figure 10: Example of an Attack Pattern in CAPEC

natural language descriptions. Each attack pattern contains an average of 268 tokens and each attack field has about 15 tokens.

2.2 ArkAngel

As indicated in Section 1.1, our thesis is the result of a joint work with Above Security. Our task was to evaluated and improve the quality of their current SIEM system, ArkAngel, a Security Information and Event Management (SIEM) system, developed in house. A SIEM is comprised of a Security Information Management (SIM) system as well as a Security Event Management (SEM) system [Dobb's, 2007]. The SIM is a software running on a computer system to collect and aggregate suspicious security logs by using users' self-defined filters as well as storing logs into a centralized repository of security events for trend analysis [Bayuk, 2007]. On the other hand, the SEM's purpose is to analyze and capture sensitive contextual information in each security event stored in a repository, such as the log of usernames, timestamps, locations, etc. [ZDNet, 2006]. These sensitive contexts provide more clear clues to security analysts in order to take defensive actions quickly.

As shown in Figure 11, the ArkAngel system first collects, selects and aggregates security logs via pre-defined filters as well as storing security events into a central repository. Then, security analysts view security events via a graphical interface, identify suspicious events, take proper actions and respond to clients in a timely fashion (within at most 2 hours in the company's case [Scarabeo et al., 2015]). However, with the increased demand for computer security, the volume of security logs has increased substantially and in order to address this issue, an intelligence layer was developed by our partner company and inserted into the ArkAngel system.

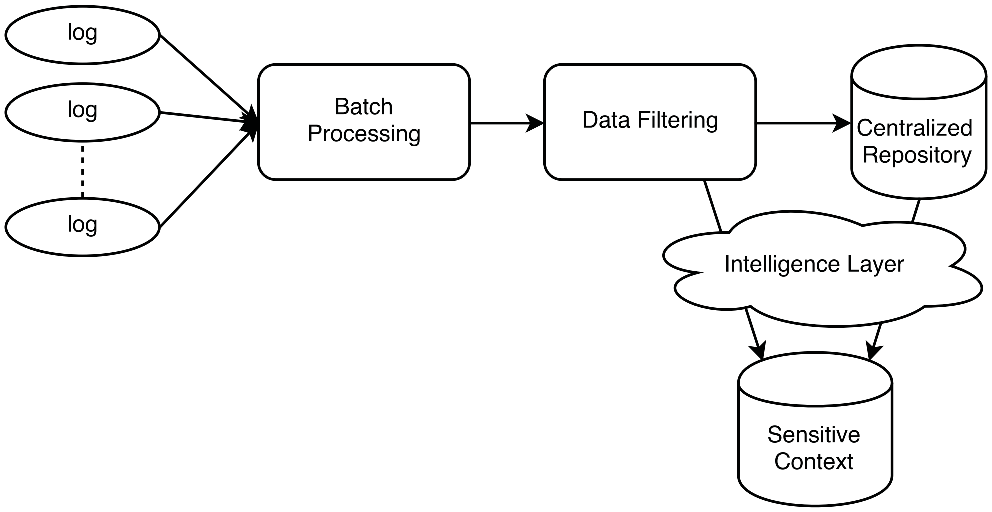


Figure 11: Workflow of ArkAngel

Figure 12 shows the workflow of the intelligence layer of ArkAngel. The correlation rules and scripts in the knowledge base are maintained daily by security analysts in order to recognize security risks in the filter process. However, with the substantial increase of security events, two problems were identified:

1. There is no standard basis for defining the pre-defined rules and scripts
   The pre-defined rules and scripts are written by security analysts based on their analysis experience. Thus, significant variations of rules and scripts exist for the same known vulnerability, which bring difficulties to security analysts analyzing suspicious events based on the short natural language descriptions. Also, without a standard basis for rule definitions, it is difficult to ensure that a new rule or a script will match new security logs.

2. High Maintenance Workload for Security Analysts
   With the increasing volume of logs, the workload of knowledge base maintenance has also increased significantly. Security analysts not only need to update the

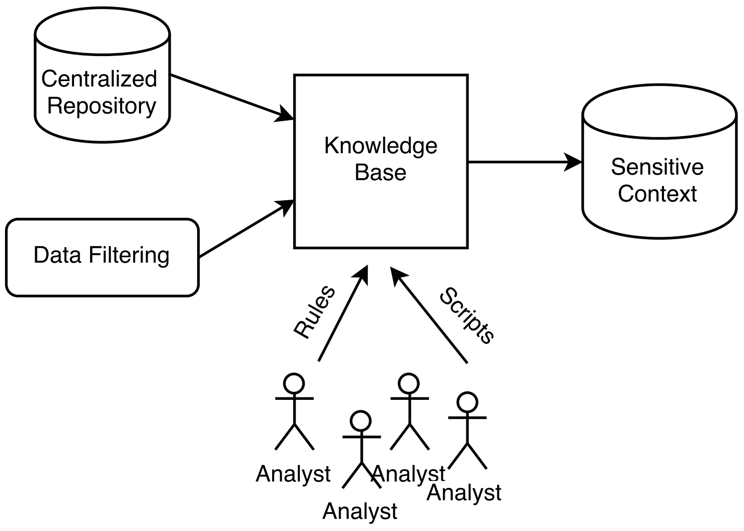


Figure 12: Workflow of the Intelligence Layer of ArkAngel

existing rules and scripts in the knowledge base, but also need to analyze larger volumes of logs which are not matched by existing rules in order to identify novel software weaknesses as well as inserting new rules into the knowledge base. Thus, the possibility of not detecting a security risk increases.

To address these two issues, security analysts realized that they needed an intelligent system able to map security events to attack patterns with standard descriptions automatically. To address this issue, a new module called *Charibdis* was developed by our partner company.

2.2.1 Charibdis

To address the two issues presented in Section 2.2, our partner company developed a new module within ArkAngel, Charibdis, which computes the semantic similarity between short snort messages and attack fields in CAPEC. Recall from Section 2.1.3 that CAPEC was developed and is maintained by MITRE Corporation (a non-profit organization) and provides detailed descriptions for each known vulnerability. Thus, the aim was that the attack fields would tackle the standard descriptions issue and the Charibdis would decrease the high workload of security analysts.

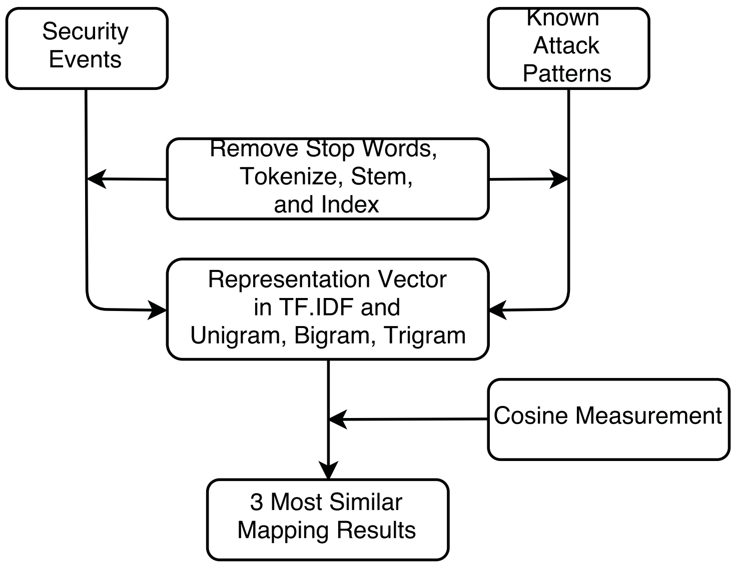

Security Events
Known Attack Patterns
Remove Stop Words, Tokenize, Stem, and Index
Representation Vector in TF.IDF and Unigram, Bigram, Trigram
Cosine Measurement
3 Most Similar Mapping Results

Figure 13: Workflow of the Charibdis System

2.2.1.1 System Overview

Charibdis[1] is a core module in the intelligence layer of ArkAngel. Its purpose is to map security events to known attack patterns in Common Attack Pattern Enumeration and Classification (CAPEC) (see Section 2.1.3) in order to reduce the workload of security analysts. Figure 13 shows an overview of the workflow of Charibdis. Charibdis takes as input security events and known attack patterns. Each security event and attack field description is first pre-processed through tokenization, removal of stop words and stemmed using the Snowball Stemmer [Porter, 2001]. Then, unigrams, bigrams and trigrams are used as terms. Document frequency (DF) and term variance (TV) are then used to filter terms with a high or low frequency. Using term frequency · inverse document frequency (TF.IDF) and the cosine measure, the similarity between snort messages and each CAPEC field is then computed. Finally, the 3 most similar CAPEC fields that have a similarity greater than a threshold Sim_T, are selected and recommended to security analysts. According to [Scarabeo et al., 2015], based on these 3 best mapped CAPEC fields, analysts can easily find vulnerability mitigations and respond to clients within 2 hours. The following describes the system in more details.

[1]In the Greek mythology, Charibdis was a sea monster that lived in the Strait of Messina and could create a huge whirlpool by swallowing huge amounts of seawater.

2.2.1.2 Inputs

The input of the Charibdis system consists of snort alert messages as *Security Events* and CAPEC attack fields as *Known Attack Patterns*.

1. Snort Alert Messages
   Figure 14 shows an example of a snort alert message taken as input. The snort alert messages are split by semi-colons into three parts. The first part (number `7540` in Figure 14) is the ID of the snort alert message in our partner's company database. The second part is the description which is composed of the snort rule name (e.g. `SERVER-OTHER` indicates that this snort alert message is about the operations in computer servers). The next words in this snort message make up the actual description (e.g. the attempt of uploading a large zip file can cause a vulnerability). This attempt has been pre-defined by security analysts in the snort framework configuration process, thus it can be captured by the snort system. The last part of the snort alert message, `"cve, 2015-2331"`, indicates that the content of this snort message is highly related to the known vulnerability in Common Vulnerabilities and Exposures (see Section 2.1.1) with the ID `2015-2331`. In the Charibdis system, only the second section (the natural language description) is used. The first part, the snort message ID, will be used in our work to evaluate the mapping results (see Chapter 3).

```
7540;"SERVER-OTHER PHP zip_cdir_new function
integer overflow file upload attempt";" cve,2015-2331"
```

Figure 14: Example of Input Snort Alert Message

2. CAPEC Attack Fields
   As described in Section 2.1.3, a CAPEC attack pattern is composed of several fields, such as the vulnerability description, exploiting steps, solutions and mitigations, etc. These fields provide detailed descriptions for a set of known vulnerabilities. Instead of using the entire attack pattern in CAPEC as the entity of known vulnerabilities, Charibdis uses individual attack fields in each attack pattern because the descriptions in an attack pattern is too general to match a specific security event and the difference in the length between snort messages and attack patterns is significant. As shown in Figure 15, one attack field contains three components separated by a semi-colon. The number `2057`

is the attack field ID given by the our partner company which is used to evaluate the mapping results (see Chapter 3); the `attack_step_description` is the title of this attack field and the rest of words describe the exploit that hackers may use to hack into the system. Once the attack fields are mapped, security analysts can obtain attack patterns and take the proper actions. As indicated in Section 2.1.3, there are 5,096 CAPEC attack fields and each field is 15 words on average.

```
2057;attack_step_description;The attacker probes for
programs running with elevated privileges.
```

Figure 15: Example of an Input Attack Field From CAPEC

2.2.1.3 Baseline Algorithm

As Figure 13 shows, snort alert messages and attack fields are pre-processed through several natural language processing techniques. Below is a brief review of each technique and an illustrative example for each step. Suppose that we have the snort messages and CAPEC attack fields below:

Snort Messages:

d_1: `BROWSER-FIREFOX Mozilla Firefox IDB use-after-free attempt`

d_2: `SQL 1 = 1 - possible sql injection attempt`

CAPEC Attack Fields:

d_3: `This category is related to the WASC Threat Classification 2.0 item SQL Injection`

d_4: `Use a browser to manually explore the website and analyze how it is constructed. Many browser's plug-in are available to facilitate the analysis or automate the URL discover`

1. Tokenization
   Given a document, the tokenization process splits the document into a sequence of tokens. In the baseline system, CAPEC fields are concatenated after the snort messages and all of them are first split by spaces and punctuation and

tokens containing numerical information (e.g. *Win32*) are removed. The remaining tokens are converted to lowercase and put into a list. In the current implementation, tokenization is made using *word_tokenize* in the Natural Language Toolkit (NLTK) [Loper and Bird, 2002] written in Python. For example, with the above documents, the tokenization result would be:

```
['browser', 'firefox', 'mozilla', 'firefox', 'idb', 'use',
'after', 'free', 'attempt']
['sql', 'possible', 'sql', 'injection', 'attempt']
['this', 'category', 'is', 'related', 'to', 'the', 'wasc',
'threat', 'classification', 'item', 'sql', 'injection']
['use', 'a', 'browser', 'to', 'manually', 'explore',
'the', 'website', 'and', 'analyze', 'how', 'it', 'is',
'constructed', 'many', 'browser', 's', 'plug', 'in', 'are',
'available', 'to', 'facilitate', 'the', 'analysis', 'or',
'automate', 'the', 'url', 'discover']
```

2. Stemming
   Word stemming is the process of obtaining the base or root of a word. Several stemmers are available publicly (e.g. the Porter Stemmer [Porter, 1980], Lancaster Stemmer [Chris et al., 1990] and Snowball Stemmer [Porter, 2001]). The baseline system uses *Snowball Stemmer* [Porter, 2001] and discards single characters (e.g. the *s* in the fourth document). After stemming, the sample documents become:

```
['browser', 'firefox', 'mozilla', 'firefox', 'idb', 'use',
'after', 'free', 'attempt']
['sql', 'possibl', 'sql', 'inject', 'attempt']
['this', 'categori', 'is', 'relat', 'to', 'the', 'wasc',
'threat', 'classif', 'item', 'sql', 'inject']
['use', 'a', browser', 'to', 'manual', 'explor',
'the', 'websit', 'and', 'analyz', 'how', 'it', 'is',
'construct', 'mani', 'browser', 's', 'plug', 'in', 'are',
'avail', 'to', 'facilit', 'the', 'analysi', 'or', 'autom',
'the', 'url', 'discov']
```

3. N-Grams Features

An n-gram is a contiguous sequence of n units. In the field of NLP, unigrams refer to single words or characters; bigrams refer to a sequence of two contiguous words or characters, trigrams refer to a sequence of three contiguous words or characters, etc. As we will see in Section 4, the baseline Charibdis system transforms all the snort messages and CAPEC attack fields into a document-by-features matrix after removing stop words (e.g. a, an, the, is, are ...) via the *CountVectorizer* method in the scikit-learn library [Pedregosa et al., 2011]. A mixture of unigrams, bigrams and trigrams is then used as features. In our example, the matrix would be:

```
     attempt  browser  inject  sql  sql inject  ...  websit analyz
d1         1        1       1    0           0  ...              0
d2         1        0       1    2           1  ...              0
d3         0        0       1    1           1  ...              0
d4         0        2       0    0           0  ...              1
```

where the columns (d_i) represent documents and the rows are the mixture of n-grams (n ranges from 1 to 3). For example, `analysi` is an unigram; `analysi autom` is a bigram and `websit analyz construct` is a trigram. In total, these four documents are converted into 85 features which are sorted in ascending alphabetic order. Each entry in this matrix is the frequency of the n-gram in the corresponding documents.

4. Term Variance

   Term Variance (TV) is used to filter out features (n-grams) that either appear too often or not often enough. TV measures how the frequency of each feature deviates from the mean. If a feature has a low variance, it means that it appears in most of the documents (the snort messages and attack fields). In this case, very few n-grams are repeated more than once, so the frequency of most features in one document (one snort message or attack field) is one. In the baseline system, the *VarianceThreshold* method in the scikit-learn library [Pedregosa et al., 2011] is used and the value of *term variance (TV)* is set to 0.98 which removes features that appear in more than 98% or less than 2% (1 - 0.98) of the documents. In our example, the features appear at least in one document (in 25% of the documents) and none exists in all of the documents. Thus, after term variance filtering, we obtain the same document-by-feature matrix. In Chapter 3, we will measure the effect of the term variance on the baseline system.

5. Document Frequency
   Document frequency (DF) refers to the number of documents which contain a specific feature. For instance, in our example, there are 4 documents in total but only 2 documents contain the specific word *happy*. Thus, the document frequency of the feature *happy* is 2 out of 4. In the baseline Charibdis system, our partner company set the *document frequency (DF) to 40* arbitrarily in the feature filtering process.

6. Term Frequency - Inverse Document Frequency
   Term frequency (TF) is the frequency of the feature which is mostly used as term weights in the document-by-term matrix. Another most widely weighting scheme is term frequency $*$ inverse document frequency ($tf * idf$) [Salton et al., 1975]. This measure represents how discriminating a term is to represent documents in a collection. The inverse document frequency is defined as $idf_i = \log(\frac{N}{df_i})$, where N is the total number of documents in the collection and df_i is the document frequency of the i_{th} term. The $tf * idf$ measures the relevance of a term to a document. For example, if a word exists in every document, its idf value is 0 and thus the $tf * idf$ is 0. On the other hand, if a word only appears in 50% of the documents, its idf is 0.3 (greater than 0) which means that this word is more discriminating. Thus, $tf * idf$ is widely used as term weights in order to represent the relevance of a term. The TF.IDF matrix in our example would be:

	attempt	browser	inject	sql	sql inject	...	websit analyz
d_1	0.30	0.30	0	0	0	...	0
d_2	0.30	0	0.30	0.60	0.30	...	0
d_3	0	0	0.30	0.30	0.30	...	0
d_4	0	0.60	0	0	0	...	0.60

7. Cosine Similarity
   Cosine similarity measures the similarity between two documents by calculating the cosine value of the angle between the two document vectors (for example, one row in the matrix above). If two documents are identical, the cosine similarity

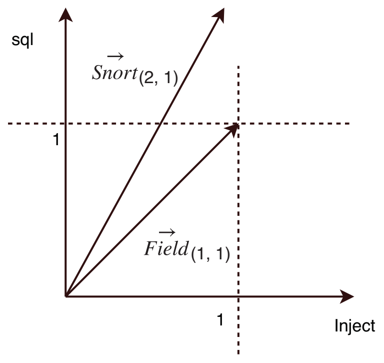


Figure 16: Example of Cosine Similarity

is 1 whereas if two documents are orthogonal, their similarity is 0. Thus, the value range of the cosine similarity is within $[0, 1]$. In this case, the normalized inner product of the snort message vectors and the CAPEC field vectors is the similarity between these two documents, which is defined as:

$$Sim_{(\vec{snort},\ \vec{field})} \;=\; \vec{snort} \cdot \vec{field} \;=\; \frac{\sum_{i=1}^{n}(w_{i\_snort} \times w_{i\_field})}{\sqrt{\sum_{i=1}^{n}(w_{i\_snort}^2)} \;+\; \sqrt{\sum_{i=1}^{n}(w_{i\_field}^2)}}$$

where w_{i_snort} and w_{i_field} are the term weights in the snort messages and the CAPEC fields and n is the number of terms in the vocabulary. In our example, suppose that the snort message d_2 and the CAPEC field d_3 are represented using only the 2 features `inject` and `sql` and we use term frequency as term weights, then their cosine similarity is:

$$Sim_{\vec{snort},\ \vec{field}} \;=\; \frac{1 \times 2 \;+\; 1 \times 1}{\sqrt{5} \times \sqrt{2}} \;=\; 0.95$$

Figure 16 shows the cosine similarity between the 2 documents represented by `inject` and `sql`.

Based on these pre-processing techniques, Charibdis reads snort messages and CAPEC fields, tokenizes them, filters unimportant features through document frequency and term variance, and transforms them into a document-by-terms matrix.

Then, each term frequency is replaced by its *TF.IDF* value calculated by the *Tfidf-Transformer* scikit-learn method. Finally the *cosine_similarity* method measures the similarity between each snort messages and attack fields and the 3 most similar CAPEC fields are returned for each snort message. These mapping results are finally recommended to security analysts.

2.3 Sentence Similarity

The workflow and the algorithm of the baseline Charibdis system is based on sentence similarity, a standard problem in natural language processing (NLP). Given two sentences, the task of measuring their similarity follows two main approaches. One is to consider only string similarity and the other is to consider the actual meaning of the sentences [Gomaa and Fahmy, 2013]. In this section, we describe related works using these these two approaches.

2.3.1 String Similarity

Computing sentence similarity at the string level is related to lexical similarity as it does not consider the meaning of the sentences. In natural language processing, two models can be used to represented input text: *character-based models* or *word-based models*.

2.3.1.1 Character-based Models

Character sequence similarity is a type of lexical similarity. A sequence of characters can be a word in natural language or a data record in a database. Since the 1960s, several approaches have been introduced to calculate character sequence similarity. Let us describe several well-known algorithms.

> [Levenshtein, 1966] describes a method called the *edit distance*. It quantifies the similarity between two character sequences by counting the minimum number of steps which are needed to transform one string into another through insertion, deletion and substitution of characters. The principle is that the fewer steps are needed to transform, the more similar the two strings are. Today, the edit

distance is wildly used in spell checking applications (e.g. *appel* is more likely to be *appeal* than *apple*).

[Allison and Dix, 1986] developed an algorithm called *longest-common-subsequence (LCS)* based on dynamic programming. The method captures the longest common character sequence from two similar strings without the consecutive restriction. For instance, given two strings "DACBGHZ" and "ACBEJQGH", the longest common subsequence is "ACBGH". Similarly, [Gusfield, 1997] proposed an algorithm to extract the *longest common substring* which is restricted by continuity conditions. In the previous example, the longest common substring is "ACB".

[Jaro, 1989] proposed a novel algorithm to measure the similarity between two strings. Given two strings S_1 and S_2, the similarity is defined as:

$$sim_j(S_1, S_2) = \begin{cases} 0 & \text{if m = 0} \\ \frac{1}{3} \times (\frac{m}{|S_1|} + \frac{m}{|S_2|} + \frac{m - t}{m}) & \text{otherwise} \end{cases}$$

where m is the number of matching characters and t is half the number of transpositions. In the Jaro distance, characters are considered matching only if they are identical and not farther than $\left\lfloor \frac{max(|S_1|,\ |S_2|)}{2} \right\rfloor$ - 1. For example, for the string S_1: *"MARTHA"* and S_2: *"MARHTA"*, the value of m, t, $|S_1|$ and $|S_2|$ are:

1. m is 6 because there are six matching characters.
2. Characters *'T'* and *'A'* need to switch position in order to match. Thus, t is 2.
3. The length of S_1 and S_2 are both six.

The Jaro similarity of these two strings is therefore: $\frac{1}{3} \times (\frac{6}{6} + \frac{6}{6} + \frac{6 - 2}{6}) =$ 0.94.

[Winkler, 1990] added a new feature to the Jaro distance measure, the prefix scale p. In [Winkler, 1990], the similarity of two string S_1 and S_2 is defined as:

$$sim_w(S_1,\ S_2) = sim_j(S_1,\ S_2)\ +\ (l\ \times\ p\ \times\ (1\ -\ sim_j(S_1,\ S_2)))$$

where sim_j is the Jaro similarity of the two input strings S_1 and S_2; l is the length of common characters at the start of the two strings and up to four characters; p is a constant scaling factor for the common prefixes score adjustment which does not exceed 0.25. Normally, the value of p is 0.1. Thus, for the previous example, the Jaro-Winkler similarity is:

$$0.94 \ + \ (3 \ \times \ 0.1 \ \times \ (1 \ - \ 0.94)) \ = \ 0.96$$

These algorithms score string similarity through the number of common characters. However, in natural language processing, words are more natural units than characters. In next section, word-based similarity metrics are introduced.

2.3.1.2 Word-Based Model

Instead matching characters, words are an important respect to consider in the similarity measurement. In this section, we describe several word-based techniques to measure string similarity.

1. [Monge et al., 1996, Monge and Elkan, 1997] proposed a simple but effective word similarity metric, called *Monge-Elkan*. The algorithm first tokenizes the input strings into tokens and scores the token pairs from each string by some internal similarity, such as the *edit distance* or the *Jaro-Winkler* similarity. Then, word pairs with the highest similarity score are picked out; their scores are summed up and normalized as the similarity of these input strings. The definition in [Monge and Elkan, 1997] is as follow:

$$sim_{Monge-Elkan}(S_1, \ S_2) \ = \ \frac{1}{|S_1|}\sum_{i=1}^{|S_1|} max\{sim'(S_{1i}, \ S_{2j})\}_{j=1}^{|S_2|}$$

   where S_1 and S_2 are two input strings, $|S_1|$ and $|S_2|$ are the number of tokens in S_1 and S_2, $sim'(S_1, \ S_2)$ is measured by the *Smith-Waterman* algorithm [Smith and Waterman, 1981]. As with the previous measures, the lower the edit distance, the more similar these strings are. Thus, the internal similarity $sim'(S_{1i}, \ S_{2j}) = 1$ - $d_{Smith-Waterman}(S_{1i}, \ S_{2j})$ 2. For instance, the *Monge-Elkan* similarity of two strings *I Love You* and *You Love Him* is:

   $sim_{Monge-Elkan}(A, \ B)$
   $= \frac{1}{3} \times (max(1 \ - \ d_{SW}(A_1, \ B_1), 1 \ - \ d_{SW}(A_1, \ B_2), 1 \ - \ d_{SW}(A_1, \ B_3)$
   $+ \ max(1 \ - \ d_{SW}(A_2, \ B_1), 1 \ - \ d_{SW}(A_2, \ B_2), 1 \ - \ d_{SW}(A_2, \ B_3)$

2Here, we use d_{SW} to represent $d_{Smith-Waterman}$

$+ \ max(1 - d_{SW}(A_3,\ B_1), 1 - d_{SW}(A_3,\ B_2), 1 - d_{SW}(A_3,\ B_3))$
$= \frac{1}{3} \times (max(1 - d_{SW}(I,\ You), 1 - d_{SW}(I,\ Love), 1 - d_{SW}(I,\ Him)$
$+ \ max(1 - d_{SW}(Love,\ You), 1 - d_{SW}(Love,\ Love), 1 - d_{SW}(Love,\ Him)$
$+ \ max(1 - d_{SW}(You,\ You), 1 - d_{SW}(You,\ Love), 1 - d_{SW}(You,\ Him))$
$= \frac{1}{3} \times (max(1 - \frac{3}{3}, 1 - \frac{4}{4}, 1 - \frac{3}{3})$
$+ \ max(1 - \frac{3}{4}, 1 - 0, 1 - \frac{4}{4})$
$+ \ max(1 - 0, 1 - \frac{3}{4}, 1 - \frac{3}{3}))$
$= \frac{1}{3} \times (0 + 1 + 1) = 0.67$

2. N-Grams

In natural language processing, *n-grams* is a sequence of n consecutive tokens. Unigrams ($n = 1$), bigrams ($n = 2$) and trigrams ($n = 3$) are widely used to represent text in a linear fashion. Many NLP applications use *n-grams* as their basic units, for example, automatic speech recognition, machine translation and similar sentences matching. [Lyon et al., 2001] indicates that the number of overlapping bigrams or trigrams derived from two or more independent texts should be quite small and most of the trigrams should only belong to the document which provides them, because English follows the principle of Zipf's law [Zipf, 2016]. Thus, if two documents contains many co-occurences of bigrams and trigrams, they are likely to be similar. [Barrón-Cedeño and Rosso, 2009] uses n-grams, with n ranging from 1 to 5 to represent two comparable documents, and the containment measure [Broder, 1997] to detect plagiarism with references. The definition of the containment measure is:

$$C(s_i \mid d) = \frac{|N(s_i)| \cap |N(d)|}{|N(s_i)|}$$

where $|N(d)|$ is the set of n-grams in the reference document and $|N(s_i)|$ is the set of n-grams in the i_{th} sentence of the suspicious document, $C(s_i \mid d)$ is the percentage of overlapping n-grams. [Barrón-Cedeño and Rosso, 2009] first splits the suspicious documents into sentences and represents each sentence by n-grams. On the other hand, the reference document is represented via n-grams directly. For each sentence in the suspicious document, if the percentage of overlapping n-grams is greater than a given threshold, the sentence is considered plagiarised from the reference documents. The best results in plagiarism detection was achieved using bigrams and trigrams because these n-grams are not only short enough to handle text meaning but also long enough to represent each sentences.

3. Vector-Space Models

   In order to calculate the sentence similarity based on the co-occurence of n-grams, the vector-space model [Salton and McGill, 1986] is often used. This model converts a document into a term-by-document matrix. Each entry in this matrix is the weight of the corresponding term. The weight of the terms can be their frequency in each document or some other weighting scheme, such as $TF.IDF$ (see Section 2.2.1.3). For instance, suppose the two sentences S_1 and S_2:
   S_1: *"This robot can speak English"*,
   S_2: *"That robot can speak French"*

   After tokenizing these two sentences into unigrams, Table 1 shows the corresponding term-by-document matrix where each entry is the term frequency in each sentence.

	S_1	S_2
This	1	0
robot	1	1
can	1	1
speak	1	1
English	1	0
That	0	1
French	0	1

Table 1: Example of a Term-by-Document Matrix

4. Similarity Metrics

   Based on the term-by-document matrix, a variety of similarity measures are widely used to measure the similarity between two sentences.

   (a) Euclidean Distance

   The Euclidean Distance is a simple distance metrics. Given two points P

and Q in the n dimensions, the Euclidean distance is defined as follow.

$$d(P, Q) \ = \ \sum_{i=1}^{n} \sqrt{(P_i \ - \ Q_i)^2}$$

where P_i and Q_i are the coordinates of points P and Q in the i_{th} dimension. In the document similarity measure, points P and Q represents two different documents and the corresponding coordinates are the term weights in the term-by-document matrix. In the previous example, the Euclidean Distance is:

$d(S_1, \ S_2)$

$= \sqrt{(This_{s_1} - This_{s_2})^2 \ + \ (robot_{s_1} - robot_{s_2})^2 \ + \ ... \ + \ (French_{s_1} - French_{s_2})^2}$

$= \sqrt{1 \ + \ 0 \ + \ 0 \ + \ 0 \ + \ 1 \ + \ 1 \ + \ 1} \ = \ 2$

(b) Jaccard Coefficient

The Jaccard Coefficient calculates the similarity of two sentences by the number of common tokens divided by the union of the tokens from the two sentences [Jaccard, 1901]. In the previous example, the Jaccard similarity is:

$$Sim_j(S_1, \ S_2) \ = \ \frac{1 \ + \ 1 \ + \ 1}{7} \ = \ \frac{3}{7} \ = \ 0.43$$

(c) Cosine Similarity

When sentences are represented as vectors, their similarity can be calculated through the cosine value of the angle of two vectors. Given two sentences S_1 and S_2, the cosine value is the product of the vectors $\vec{S_1}$ and $\vec{S_1}$ divided by the normalized lengths of two vectors. The definition is:

$$Sim_{cosine}(\vec{S_1}, \vec{S_2}) \ = \ \frac{\vec{S_1} \cdot \vec{S_2}}{|S_1||S_2|}$$

where $\vec{S_1}$ and $\vec{S_2}$ are n dimensional vectors over the term set $T \ = \ \{t_1, \ t_2, \ ..., \ t_n\}$. Each value of the vector is the term frequency or any other term weight. For instance, if we use term frequency in the previous example, the cosine similarity is:

$Sim_{cosine}(\vec{S_1}, \vec{S_2}) = \frac{1 \ \times \ 0 \ + \ 1 \ \times \ 1 \ + \ ... \ + \ 0 \ \times \ 1}{\sqrt{5} \ \times \ \sqrt{5}} = \frac{3}{4} \ = \ 0.6$

The *Cosine Similarity* is independent of document length, because it normalizes the length of each document vector [Huang, 2008]. For example, if unigrams are used to represent these two sentences

S_1: *I like computer science*

S_2: *computer science is my major*

the cosine similarity is 0.52. If we extend S_1 to *I like computer science I like computer science* and keep S_2 as before, the similarity is also 0.52 even though the S_1 is much longer. Thus, the *Cosine Similarity* is widely used in information retrieval.

Lexical similarity measures work very well when two documents contain many common lexical units. In addition, in order to improve their precision, many preprocessing methods are used, such as stemming, stop-words removal, using different term weights. However, these lexical-based metrics do not take into account the meaning of words. For instance, the similarity between these two sentences *I own a dog* and *I have an animal* cannot be found by lexical similarity [Mihalcea et al., 2006].

2.3.2 Semantic Similarity

Compared to lexical similarity, semantic similarity takes into account the meaning of two sentences. This is done through two approaches: knowledge-based and corpus-based models [Gomaa and Fahmy, 2013].

2.3.3 Knowledge-Based Models

Knowledge-based models make use of taxonomies of concepts. By quantifying the degree of semantic similarity of words and concepts derived from publicly available resources, for example *WordNet* [Miller, 1995], the relatedness between words can be measured [Mihalcea et al., 2006]. Note that the concept of *semantic relatedness* is more general than semantic similarity as it includes similarity and dissimilarity [Budanitsky and Hirst, 2006]. In this section, we only consider semantic similarity and present several word-to-word similarity approaches based on WordNet. Finally, an approach measuring sentence semantic similarity based on the aggregation of word-to-word scores is described.

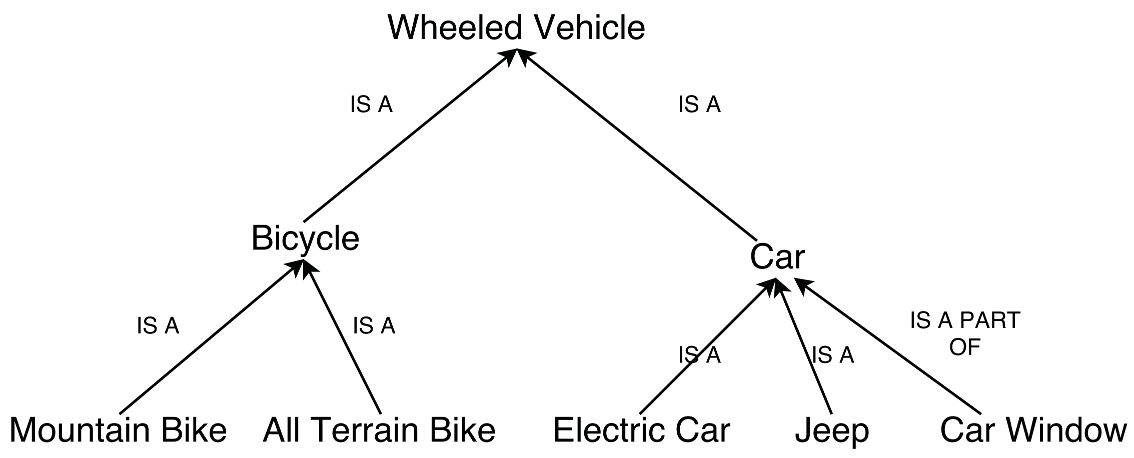


Figure 17: A Fragment of WordNet Hierarchy

2.3.3.1 WordNet

WordNet is a large lexical database developed by George Miller and colleagues at Princeton University [Miller, 1995]. It includes nouns, verbs, adjectives and adverbs which are organized in taxonomic hierarchies. From the root to the leaves, the sense of words is increasingly specific. The basic unit in WordNet is called a *synset* which groups semantic related words into distinct sets. In total, there are 117,000 synsets interlinked with each other via several types of semantic relations. Figure 17 shows a fragment of the WordNet hierarchical levels. In nouns and verbs synsets, the *hyperonymy / hyponymy* (also called is-a) relation and the *meronymy / holonymy* (also called is-a-part-of) relation are most frequently used. As an example, *Electric Car* is a *Car* and *Car Window* is a part of *Car* (see Figure 17). In addition, the relations in adjective synsets induce *synonymy / antonymy* (e.g. beautiful and attractive, wet and dry) and only a few adverbs are organized in WordNet which are derived from the adjectives morphological affixation. One WordNet restriction is that synset relations are only defined for words with the same part-of-speech. [Miller, 1995]

2.3.3.2 Word-To-Word Semantic Similarity

Based on the WordNet public lexical resource, several word-to-word similarity approaches have been proposed [Pedersen et al., 2004].

[Wu and Palmer, 1994] calculate the similarity between two words in the WordNet taxonomy through the depth of the least common subsumer (LCS). This is defined as:

$$Sim_{wup}(w_1,\ w_2)\ =\ \frac{2\ \times\ depth(LCS)}{depth(w_1)\ +\ depth(w_2)}$$

where w_1 and w_2 are two words and the depth value is the number of nodes between a word and the least common subsumer. For instance, the similarity between *Mountain Bike* and *Jeep* in Figure 17 is:

$$Sim_{wup}(Mountain\ Bike,\ Jeep)\ =\ \frac{2\ \times\ 1}{3\ +\ 3}\ =\ 0.33$$

[Resnik, 1995] argues that evaluating semantic similarity through counting the nodes in WordNet should not be used, because the real distance between two linked nodes varies widely. For example, *rabbit ears* is a *hyponymy* of *television antenna* in WordNet. To address this problem, Resnik proposes an alternative semantic similarity metric in WordNet. He first explains the definition of *information content*.

$$IC(w_1)\ =\ -\log(P_{w_1})$$

where P_{w_1} is the probability of encountering w_1 in WordNet. [Resnik, 1995] argues that the probability of encountering a hyperonym is larger than a hyponym because the meaning of a hyperonym is more general. If only one root node exists in the taxonomy, then P_{root} is 1. Based on the information content, the definition of the similarity between two words is:

$$Sim_{res}(w_1,\ w_2)\ =\ IC(LCS(w_1,\ w_2))$$

Based on the information content and Resnik's approach, [Jiang and Conrath, 1997] defines word-to-word similarity as:

$$Sim_{jiang}(w_1,\ w_2)\ =\ \frac{1}{IC(w_1)\ +\ IC(w_2)\ -\ 2\ \times\ IC(LCS(w_1,\ w_2))}$$

One year later, [Lin et al., 1998] evaluates concept similarity in WordNet based on the information content:

$$Sim_{lin}(w_1,\ w_2)\ =\ \frac{2\ \times\ IC(LCS(w_1,\ w_2))}{IC(w_1)\ +\ IC(w_2)}$$

[Leacock and Chodorow, 1998] define the similarity between two words via the shortest path between them in the WordNet hierarchy, normalized by twice the maximum depth of these two words.

$$Sim_{lch}(w_1,\ w_2)\ =\ -\log(\frac{length}{2\ \times\ D})$$

where w_1 and w_2 are two comparison words, *length* is the shortest path between them and D is the maximum depth of these two words in WordNet. In the previous example, the similarity is:

$$Sim_{lch}(Mountain\ Bike,\ Jeep)\ =\ -\log(\frac{5}{2\ \times\ 3})\ =\ 0.08$$

The availability of lexical resources makes the computation of word-to-word similarity metrics straightforward; whereas text-to-text similarity measure still mainly relies on the conventional vector-space models (see Section 2.3.1.2). [Mihalcea et al., 2006] introduces an approach which measures text similarity based on word-to-word similarity scores. Given two input text fragments T_1 and T_2, these two texts are tokenized into words after removing punctuation and stop words. Then, each word w_1 in segment T_1 is compared with every word that shares the same part-of-speech in T_2 in order to seek a word w_2 which has the maximum similarity with word w_1 through a word-to-word similarity approaches (see Section 2.3.3.2). In this step, adjectives and adverbs whose similarity cannot be obtained from WordNet (see Section 2.3.3.1) are evaluated via identical occurrence. Next, the same process is used for each word in T_2 to determine the maximum similar word in T_1. Finally, each word similarity in each text fragment is weighted by the word *idf* value, summed up and normalized by the sum of word idf values in each fragment. The formulation is:

$$sim(T_1,\ T_2)\ =\ \frac{1}{2}((\frac{\sum_{w\in T_1} maxSim(w,\ T_2) * idf(w)}{\sum_{w\in\{T_1\}} idf(w)}) + (\frac{\sum_{w\in T_2} maxSim(w,\ T_1) * idf(w)}{\sum_{w\in\{T_2\}} idf(w)}))$$

where $idf(w)$ is derived from the British National Corpus (a 100 million words corpus) [Clear, 1993]. The similarity score is between 0 and 1, where 1 indicates the two text segments are identical. To evaluate this measure, the Microsoft paraphrase corpus [Dolan et al., 2004], which contains 5,801 tagged text pairs, were evaluated and achieved F-measure of 81.3% compared to the vector-space model of 75.3%.

2.3.4 Corpus-Based Model

Vector-space models rely on words co-occurrences to determine the similarity score between texts while knowledge-based models depend a knowledge database in order to compute semantic similarity. However, corpus-based models extract similar words and topics from large corpora. One of the most popular semantic similarity techniques in corpus-based model is *Latent Semantic Analysis* [Landauer and Dumais, 1997].

2.3.4.1 Latent Semantic Analysis

The basic idea of latent semantic analysis (LSA) is that semantically similar words and phrases will occur in similar meaning contexts [Gomaa and Fahmy, 2013]. Thus, LSA focuses on weighting words based on their importance for each potential meaning (also called latent topics) in a large corpus. Given a term-by-document matrix which represents a corpus, LSA tries to convert the matrix to a term-by-latent topics matrix using singular value decomposition (SVD). SVD is a well-known matrix decomposition method. The principle of SVD is that any rectangular matrix can be decomposed into the product of three matrices:

$$C = U\Sigma V^T$$

where C is the original term-by-document matrix, U and V^T are two orthogonal matrices and any two distinct rows in matrix U are orthogonal and any two columns in matrix V^T are orthogonal, Σ is a square diagonal matrix and each value in the diagonal are eigenvalues, which indicate how important this latent topic is. Unimportant latent topics (with lower eigenvalues) will be ignored and matrix dimension will be reduced. Instead of discarding the words in the omitted dimensions, SVD projects these words to the saved dimensions which highlights the text similarity and distinguish unrelated corpora [Schütze et al., 2008]. For example, suppose three sentences are as following:

S_1: *Machine learning is super fun.*

S_2: *Python is super super cool.*

S_3: *Data science is fun.*

First, these sentences are converted into a term-by-document matrix after removing stop words. Each entry in the matrix is the term frequency in the sentences. Thus, we can have the following normalized matrix.

	S₁	**S₂**	**S₃**
cool	0	0.4	0
data	0	0	0.6
fun	0.5	0	0.6
learning	0.5	0	0
machine	0.5	0	0
python	0	0.4	0
science	0	0	0.6
super	0.5	0.8	0

After we decompose it and reduce the dimension from 8 (8 features) to 2, the normalized latent topic matrix is:

	S₁	**S₂**	**S₃**
cool	0.2	0.3	0
data	0.2	0	0.5
fun	0.5	0.1	0.6
learning	0.3	0.2	0.1
machine	0.3	0.2	0.1
python	0.2	0.3	0
science	0.2	0	0.5
super	0.7	0.8	0

In this example, the default number of latent topics is eight because there are eight features in these three sentences. After we reduced the dimensions to two, several unnecessary topics are collapsed and the similarity between two similar sentences is highlighted. If we measure the similarity between S_1 and S_2 in both the term-by-document matrix and the latent topics matrix, we can easily observe the similarity enhancement provided by LSA.

$$\begin{aligned}
&Sim_{default}(S_1,\ S_2)\\
&= 0 \times 0.4 \ + \ 0 \times 0 \ + 0.5 \times 0 \ + \ 0.5 \times 0 \ + \ 0.5 \times 0 \ + \ 0 \times 0.4\\
&\quad + \ 0 \times 0 \ + 0.5 \times 0.8 \ = \ 0.4\\
&Sim_{latent}(S_1,\ S_2)\\
&= 0.2 \times 0.3 \ + \ 0.2 \times 0 \ + 0.5 \times 0.1 \ + \ 0.3 \times 0.2 \ + \ 0.3 \times 0.2\\
&\quad + \ 0.2 \times 0.3 \ + \ 0.2 \times 0 \ + 0.7 \times 0.8 \ = \ 0.85
\end{aligned}$$

As we can see, although there are only two common words between S_1 and S_2 (*super* and *is*), the similarity of these two sentences has been doubled after using LSA because both sentences express similar meaning.

2.3.4.2 Other Similarity Measures

In addition to the latent semantic analysis, other semantic similarity approaches have been proposed. Based on large corpora, term pairwise similarity matrix and relevant documents can be extracted and used in measuring semantic similarity.

1. [Royer, 2005] proposed an approach called *Generalized Latent Semantic Analysis (GLSA)* which is based on latent semantic analysis and the use of large corpora. Given a collection of documents C with vocabulary V and a large corpus W, a term-by-document matrix D is constructed based on the documents C. Then, a word-by-word pairwise similarity matrix is build for vocabulary V using the point-wise mutual information measure [Manning and Schütze, 1999] from the large corpus W. Each entry in the matrix is the similarity weight between a word in the row and a word in the column. Next, the matrix U^T is obtained through reducing the pairwise matrix dimensions to k by SVD decomposition (see Section 2.3.4.1). Finally, the document vector $\hat{D}$ is calculated through the product of the U^T matrix and the term-by-document D ($\hat{D} = U^T D$). Columns in $\hat{D}$ represent documents in k dimensions which can be used to measure documents similarity using conventional metrics. Compared with LSA (see Section 2.3.4.1), GLSA focuses on the operation of word-by-word pairwise matrix and measures similarity over the document vectors matrix.

2. Traditional similarity metrics cannot perform well with short text snippets. For example, the cosine similarity between *artificial intelligence* and AI is zero even though their meaning is the same. To address this issue, [Sahami and Heilman, 2006] introduced a novel method, called kernel function, where the similarity between these two phrases is computed as 83.1%. Assume that the short snippet is x, and the procedure is:

   (a) Search for x using the Google search engine.

   (b) For each retrieved document (at most n documents), extract *context descriptions* of x and build context vectors v_i with $TF.IDF$ weight for each description.

(c) Truncate each vector by selecting the m highest TF.IDF weight terms.

(d) Calculate the centroid of $C(x)$ the L_2 normalized context vectors:

$$C(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{v_i}{||v_i||_2}$$

(e) Finally, replace the short snippets by the normalized centroid:

$$x_{new} = \frac{C(x)}{||C(x)||_2}$$

In addition, several ad-hoc parameters are used. In order to obtain precise $TF.IDF$ value, [Sahami and Heilman, 2006] computed documents N and df from a large sample of web documents which are not limited to the n retrieved documents in the algorithm. For the *context description* extraction, a 1,000 characters (the total number of token characters) window centered at the original snippets is used. In the terms selection process, setting $m = 50$ can result in a good trade-off between similarity value and program efficiency.

In character-based models, all the algorithms measure similarity by calculating the number of common characters between two words or steps used to change one word to another. Similarly, when measuring the similarity between two sentences, the number of common tokens are used. These tokens can be represented in many ways, such as n-grams, words, etc. The advantage of character-based models and word-based models is that they are easy to understand and use efficiently; however these models cannot measure the semantic similarity between sentences. In addition, by using statistics and analyzing large quantities of corpora, several algorithms (e.g.: LSA, GLSA) are able to estimate the semantic similarity between sentences. However, unlike string similarity methods, these algorithms are not straightforward and need corpora.

In this chapter, we have presented an overview of previous work on the three widely used public vulnerability repositories. We then described the workflow of the current Security Information and Event Management (SIEM) system and the intelligence component, Charibdis. Finally, in Section 2.3, we described the most common similar sentence mapping in terms of string similarity and semantic similarity. In Chapter 3, we will describe the evaluation metric used to evaluate the Charibdis system.

Chapter 3

Baseline System Evaluation

In [Scarabeo et al., 2015], the baseline Charibdis system (see Section 2.2.1) was only evaluated in terms of coverage. This measure, called *Mapping Rating*, evaluated the number of snort messages that were matched to at least one CAPEC field. In Section 3.1, we describe this initial evaluation metric and in Section 3.2 we explain how we evaluated the quality of the recommended fields in terms of precision, recall and F-Measure.

3.1 Initial Evaluation

As described in Section 2.2.1.3, the information provided in an attack pattern is too general to be mapped to a specific security event and the difference in length between security events (8 tokens on average) and attack patterns (268 tokens on average) is quite significant, thus attack fields are mapped to security events in the baseline[1] system. Recall from Section 2.2.1, that because the average length of security events is 8 tokens, not much information can be extracted from them to be used in building an automated model for detecting the attacks. In addition, CAPEC attack fields have on average 15 tokens, twice as many tokens than security events. In order to supplement more information for security events and equalize the length of snort messages and attack fields, two security experts in our partner company wrote descriptions for each snort rule name (see Section 2.2.1.1). In this section, we first describes the snort rule name expansion followed by the introduction of the mapping rate.

[1]We use the term *baseline* to refer to the original Charibdis system (see Section 2.2.1).

3.1.1 Snort Rule Name Description

To overcome the issue of short snort messages, security experts at Above Security analyzed 32,246 different snort alert messages and identified 68 unique snort names. For example, Table 2 shows two messages that share the same snort name (*INDICATOR-SCAN*). For each snort name, two analysts wrote a description of about 7 words. Table 3 shows the expansion of 4 such terms. By replacing snort names in the original snort messages with their longer descriptions, the length of each snort message increased from an average of 8 words to an average of 15 words.

Snort ID	**Snort Content**
6790	INDICATOR-SCAN inbound probing for IPTUX messenger port
16800	INDICATOR-SCAN cybercop os PA12 attempt
796	FILE-IDENTIFY JPG file attachment detected
28342	SERVER-WEBAPP Oracle iSQLPlus username overflow attempt

Table 2: Example of Same Rule Name

Rule Name	**Expanded Description**
INDICATOR-SCAN	Indications of scanning in network traffic
FILE-IDENTIFY	File extension file magic or header found in the traffic
SERVER-WEBAPP	Web based applications on servers
FILE-FLASH	Flash files

Table 3: Example of Snort Rule Name Expansion

3.1.2 Mapping Rate

In [Scarabeo et al., 2015], Charibdis was evaluated based on the mapping rate of 32,246 snort alert messages (see Figure 14) and 5,096 CAPEC attack fields (see Figure 15). As shown in Figure 18, 32,246 snort messages were first extended by the snort rule name description (see Section 3.1.1). As indicated in Section 2.2.1.3, all of the attack fields were tokenized, stemmed, and indexed into a document-by-term matrix. After removing features whose document frequency (DF) is lower than 40, the number of features, a mixture of unigrams, bigrams and trigrams, decreased from 192,586 to 2,213. Thus, the shape of the document-by-term matrix is 37,342 $\times$ 2,213 where 37,342 is the total number of snort messages and attack fields (32,246 + 5,096

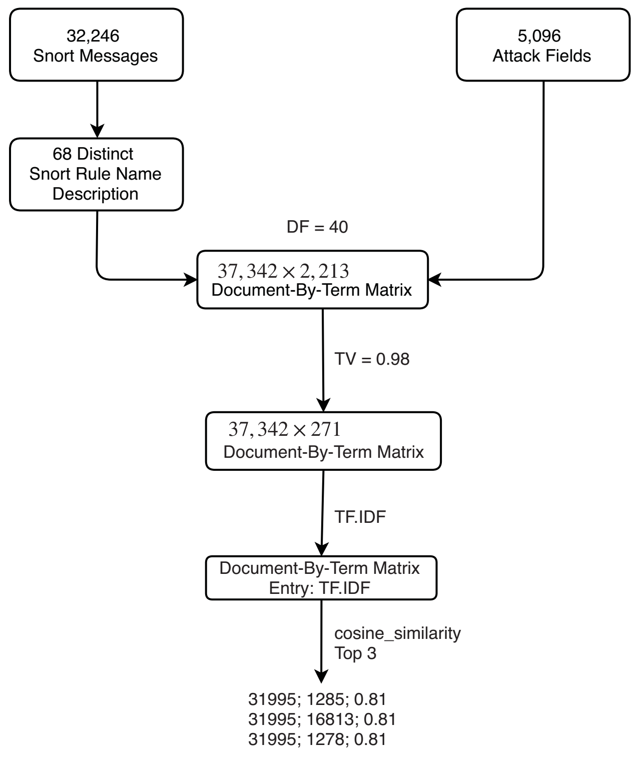


Figure 18: Detailed Workflow of the Original Evaluation of the Baseline System

= 37,342). Then, the term variance (TV) filtered out high and low frequency features and only 10% of the n-grams were kept. Finally, term frequencies are replaced by TF.IDF values in each matrix entry, the *cosine_similarity* measured the similarity between each snort message and each attack field and the 3 most similar fields were chosen as results. Figure 18 illustrates an example of snort message number 31,995 mapped to CAPEC fields number 1,285, 1,278 and 16813 respectively with a cosine measure of 0.81. The context of these four documents are given as follows.

Snort Messages

31995: *FILE-FLASH Adobe Flash Player movie signed integer memory corruption attempt*

CAPEC Fields

1285: *Use an automated tool to record the variables passed to a flash file.*

1852: *Using a browser or an automated tool an attacker records all instances of HTML documents that have embedded Flash movies. If there* *is an embedded Flash movie he lists how to pass global parameters to the Flash movie from the embedding object.*

1278: *Use an automated tool to record all instances of URLs which have embedded Flash movies and list the parameters passing to the Flash movie*

In baseline system, all CAPEC fields with a positive similarity ($Sim_{MIN} > 0$) with a snort message S were considered as valid mappings for message S. Otherwise, S is not matched to any field. To measure the system, [Scarabeo et al., 2015] used a metric called *mapping rate (MR)* which measures the number of snort messages that were matched to at least one CAPEC field out of the total number of snort messages.

$$MR = \frac{Number\ of\ Matched\ Snort\ Messages}{Total\ Snort\ Messages}$$

In their experiments with the baseline system, the number of snort messages that were matched to at least one attack field is 31,906. Thus, the *mapping rate (MR)* of the baseline system was calculated as:

$$MR = \frac{31,906}{32,246} = 98.94\%$$

Although the MR is high, the quality of the recommended CAPEC fields was not evaluated. In the next section, we describe how we evaluated the quality of the baseline's system's mapping.

3.2 Gold-Standard Evaluation

3.2.1 Gold-Standard

To evaluate the mapping quality, we created a *gold-standard* dataset by asking two cyber security experts to evaluate the quality of the output of the system. The gold-standard contains the mapping of 3,165 snort messages mapped to at most 6 CAPEC fields. This gave rise to 16,826 tagged mappings. Each mapping was annotated by the two experts with one of three levels of quality:

1. Correct
   If the security analysts could use the mapped CAPEC field directly as a solution, this mapping result was evaluated as *correct*. In the gold-standard dataset, the correct mappings were labelled as *'1'*.

2. Acceptable
   If the security analysts could generate a client solution by referring to the mapped CAPEC field, this mapping result was considered as *acceptable*. In gold-standard dataset, the acceptable mappings were tagged as *'0.5'*.

3. Incorrect
   If the mapped CAPEC field was not useful, this mapping result was labelled as *incorrect*. *'0'* was used to represent the incorrect mappings in the gold-standard dataset.

Table 4 shows three examples of the different mapping quality annotations in the gold-standard dataset. The *Snort Message ID* and *CAPEC Field ID* are the sequence number used to evaluate the mapping result (see Section 2.2.1.2). For instance, the second example indicates that snort message number 36 was mapped to CAPEC field 16,506 and was judged as acceptable by the security experts. Table 5 shows statistics of the gold-standard. As the table shows, 9,222 mappings were labelled as *correct*; 5,496 mappings were tagged as *acceptable* and 2,108 mappings were judged *incorrect*. Based on the gold-standard dataset, several metrics were used.

Snort Message ID	**CAPEC Field ID**	**Mapping Quality**
167	16005	'1'
36	16506	'0.5'
496	16564	'0'

Table 4: Extract of the Gold-Standard Dataset

3.2.2 Evaluation Metrics

The output generated by the baseline system was evaluated against the gold-standard (see Section 3.2.1). As shown in Figure 5, the gold-standard only contains the evaluation of 16,826 mappings. However, with 32,246 snort messages mapped to a maximum

Tag	**Number of Mappings**
Correct	9,222
Acceptable	5,496
Incorrect	2,108
Total	**16,826**

Table 5: Statistics of the Gold-standard Built by Two Cyber Security Analysts

of 6 CAPEC fields, the total number of possible mappings is 193,476 (6 $\times$ 32,246), therefore the evaluation of the outputs of the baseline system was made only on the overlapping answers; mappings provided by the system that were not included in the gold-standard were therefore not evaluated. For each overlap, three measures were recorded: *Correct Mapping*, *Acceptable Mapping* and *Incorrect Mapping*, depending on how the mapping quality was judged in the gold-standard dataset. Following the advice of our security analysts, recall was deemed more important than precision. Indeed, in this domain, it is preferable to alert clients too often with false alarms than to miss potential cyber threats. To account for this, two types of precision were computed: strict precision (P^S) and lenient precision (P^L) which are defined as:

$$\textit{Strict Precision: } P^S = \frac{Correct\ Mappings}{(Correct\ +\ Acceptable\ +\ Incorrect)\ Mappings}$$

$$\textit{Lenient Precision: } P^L = \frac{(Correct\ +\ Acceptable)\ Mappings}{(Correct\ +\ Acceptable\ +\ Incorrect)\ Mappings}$$

as well as two types of recall: strict recall (R^S) and lenient recall (R^L):

$$\textit{Strict Recall: } R^S = \frac{Correct\ Mappings}{Correct\ +\ Acceptable\ +\ Incorrect}$$

$$\textit{Lenient Recall: } R^L = \frac{(Correct\ +\ Acceptable)\ Mappings}{Correct\ +\ Acceptable}$$

Finally, we also calculated a series of F-Measures, which are a weighted combination of precision and recall. F-Measure is defined as $F_\beta = \frac{(\beta^2\ +\ 1)\ \times\ P\ \times\ R}{\beta^2\ \times\ P\ +\ R}$. If $\beta = 1$, then precision and recall have the same importance; if $\beta < 1$, it means that recall is favored; if $\beta > 1$, then precision is more important. In these experiments, we set the weight beta to 0.5 ($F_{0.5}$), 1 (F_1) and 2 (F_2) respectively and also computed two versions: lenient F-Measures and strict F-Measures. These F-Measures are defined as:

$$Strict\ F_{0.5}^{S} = \frac{(0.5^2\ +\ 1)\ \times\ P^S\ \times\ R^S}{0.5^2\ \times\ P^S\ +\ R^S} = \frac{1.25P^SR^S}{0.25P^S\ +\ R^S}$$

$$Lenient\ F_{0.5}^{L} = \frac{(0.5^2\ +\ 1)\ \times\ P^L\ \times\ R^L}{0.5^2\ \times\ P^L\ +\ R^L} = \frac{1.25P^LR^L}{0.25P^L\ +\ R^L}$$

$$Strict\ F_{1}^{S} = \frac{(1^2\ +\ 1)\ \times\ P^S\ \times\ R^S}{1^2\ \times\ P^S\ +\ R^S} = \frac{2P^SR^S}{P^S\ +\ R^S}$$

$$Lenient\ F_{1}^{L} = \frac{(1^2\ +\ 1)\ \times\ P^L\ \times\ R^L}{1^2\ \times\ P^L\ +\ R^L} = \frac{2P^LR^L}{P^L\ +\ R^L}$$

$$Strict\ F_{2}^{S} = \frac{(2^2\ +\ 1)\ \times\ P^S\ \times\ R^S}{2^2\ \times\ P^S\ +\ R^S} = \frac{5P^SR^S}{4P^S\ +\ R^S}$$

$$Lenient\ F_{2}^{L} = \frac{(2^2\ +\ 1)\ \times\ P^L\ \times\ R^L}{2^2\ \times\ P^L\ +\ R^L} = \frac{5P^LR^L}{4P^L\ +\ R^L}$$

Table 6 shows the default parameters indicated in Section 3.1.2 that we used to evaluate the baseline system. Sim_{MIN} represents the minimum similarity threshold to match messages. With $Sim_{MIN} = 0$, this means that as long as the snort message and attack field are not completely orthogonal, they are considered similar. *Expansion* indicates the use of snort rule name description to extend snort messages (see Section 3.1.1). As Table 7 shows, the number of acceptable mapping is quite high as it accounts for 94% (5,178 / 5,496) of the total acceptable mappings, whereas only 1% of the correct mappings were found. The P^L was 97.96% because of the contribution of acceptable mappings while the R^L was only 35.22%. Table 8 shows that the $F_{0.5}^L$ and F_1^L were 72.23% and 51.81% respectively.

System	**Sim$_{\mathbf{MIN}}$**	**DF**	**TV**	**Expansion**	**Nb of Features**
Baseline	0	40	0.98	Yes	140

Table 6: Description of Input Parameters in Baseline System

System	**Number of Mappings**			**Lenient**		**Strict**	
	Correct	**Acceptable**	**Incorrect**	**P$^{\mathbf{L}}$**	**R$^{\mathbf{L}}$**	**P$^{\mathbf{S}}$**	**R$^{\mathbf{S}}$**
Baseline	108	**5,178**	6	**97.96%**	**35.22%**	0.11%	0.07%

Table 7: Precision and Recall of the Baseline System

As we can see, although the mapping rate is 98.94%, the mapping quality is low because only 1% of the correct mappings were found. In next three chapters, we will describe several approaches to address this problem.

System	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Baseline	**72.23%**	**51.81%**	40.40%	0.09%	0.08%	0.07%

Table 8: F-Measure of the Baseline System

In this chapter, we have described the workflow of the baseline system and the attempt of [Scarabeo et al., 2015] to improve it through snort rule expansion. In addition, we explained how the mapping rate was initially evaluated (see Section 3.1.2) and how the measurement did not measure the quality of the mapping. We then described our work to evaluate the quality of the baseline's output by creating a gold-standard and using the standard metrics of precision, recall and F-measure. In order to enhance the performance of the baseline system, the next chapters investigate three approaches:

1. Feature Selection and Snort Messages Supplement.
2. Pre-clustering Snort Messages.
3. Semantic Mapping by Latent Semantic Analysis.

In the next chapter, we will provide a detailed description of the snort messages supplement methodology as well as an analysis of the evaluation of the outputs.

Chapter 4

Feature Selection and Snort Messages Supplement

Table 7 in Chapter 3 showed that the recall of the baseline system was only 35%. In order to improve the system performance, we experimented with three approaches:

1. Feature Selection and Snort Messages Supplement.

2. Pre-clustering Snort Messages.

3. Semantic Mapping by Latent Semantic Analysis.

In this chapter, we describe the first approach: n-grams feature selection to analyze the feature distribution and snort messages supplement. Section 4.1 describes our experiments with the use of a variety of feature sets and their effect on the evaluation of the system. After analyzing the feature distribution, we noticed that many snort messages suffered from a sparse representation. Indeed, although the snort rule descriptions extend the length of original snort messages (see Section 3.1.1), most of these messages are still quite short (below 15 words). To address this issue, we investigated the use of entities in the Common Vulnerabilities and Exposures (CVE) (see Section 2.1.1) to further supplement snort messages (see Section 4.2). The effect of this strategy is analyzed in Section 4.2.3.

4.1 Feature Selection

In the baseline system (see Chapter 3), snort messages and CAPEC fields are represented by a mixture of unigrams, bigrams and trigrams. However, the contribution of each type of n-gram was not clear. To measure the usefulness of each type of n-gram, three experiments were performed: the use of unigrams only, bigrams only and trigrams only. Sections 4.1.1, 4.1.2 and 4.1.3 describe these experiments; while Section 4.1.4 provides an overall evaluation.

4.1.1 Experiments Based on Unigrams

Following the workflow of the baseline system (see Section 3.1.2), after removing stop words and numbers, each snort message and CAPEC field were stemmed and tokenized. Unigrams were used as the only features to represent each snort alert messages and CAPEC fields. The document frequency (DF) and term variance frequency (TV) were then used to filter terms. Using various values for the parameters in baseline system, we ran eight experiments (see Table 9). These experiments gave rise to a feature set from size ($|F|$) of 111 to 15,963 unigrams. These features were then represented by their TF.IDF values in the document-by-term matrix whose size was (32,246 + 5096) $\times|F|$. Finally, the cosine measure was used to calculate the distance between snort messages and CAPEC fields, choosing the nearest 3 fields as mapping results.

Exp.	**Sim$_\mathbf{T}$**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	40	0.98	Yes	135
Exp. #2	0	0	0.98	Yes	135
Exp. #3	0	40	0	Yes	892
Exp. #4	0	0	0	Yes	15,963
Exp. #5	0	40	0.98	No	99
Exp. #6	0	0	0.98	No	99
Exp. #7	0	40	0	No	882
Exp. #8	0	0	0	No	13,709

Table 9: Description of the Unigram Experiments

Tables 10 and 18 in Appendix A show details of the results; while Figure 19 summarises the results graphically. As Figure 19 shows, the metrics values in the snort

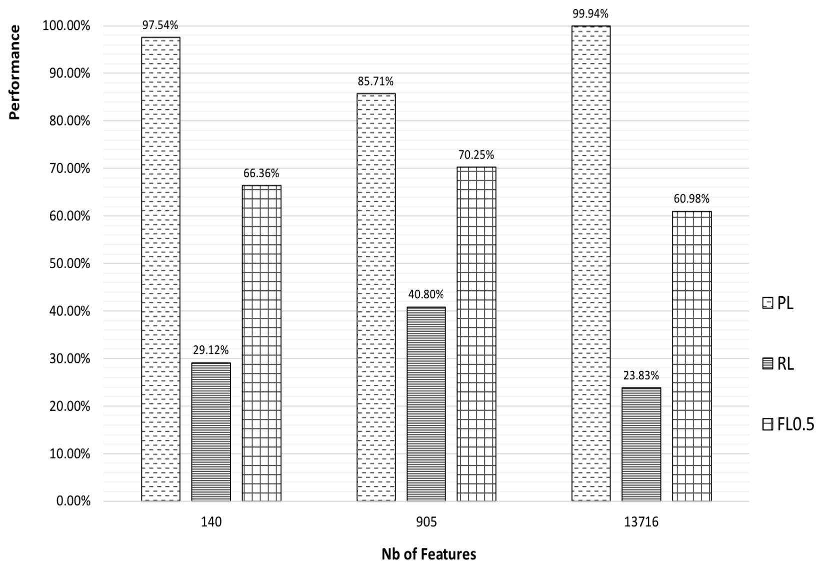


Figure 19: Results of Unigram Experiment

rule extension scenario, where the R^L was 40.80% with 905 features (DF = 40) followed by the 29.12% with 140 features (TV = 0.98). Although the P^L reached 99.94% with 13,716 unigrams (without filters), its R^L was only 23.83% which is the lowest value among these three experiments. Similarly, 70.25% was the highest value of $F^L_{0.5}$ with 905 features whereas 60.98% was the smallest. In contrast, the best $F^L_{0.5}$ was 39.22% without snort rule name expansion (see Table 10), thus we will only focus on the expansion scenario in the following experiments.

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**66.36%**	44.85%	33.87%	0.11%	0.09%	0.07%
Exp. #2	**66.36%**	44.85%	33.87%	0.11%	0.09%	0.07%
Exp. #3	**70.25%**	**55.28%**	**45.58%**	**23.44%**	**21.52%**	**19.89%**
Exp. #4	**60.98%**	38.48%	28.11%	38.46%	28.10%	22.14%
Exp. #5	**0.84%**	0.34%	0.21%	0.59%	0.24%	0.15%
Exp. #6	**0.84%**	0.34%	0.21%	0.59%	0.24%	0.15%
Exp. #7	**3.59%**	1.48%	0.93%	5.09%	2.13%	1.35%
Exp. #8	**39.22%**	20.52%	13.90%	3.20%	1.87%	1.32%

Table 10: F-Measures of the Unigram Experiments

4.1.2 Experiments Based on Bigrams and Trigrams

The next set of experiments focused on the use of bigrams and trigrams only. Using the snort rule name descriptions, bigrams are used as the only features to represent each snort message and CAPEC field and the algorithm pipeline is identical as with unigrams. Various parameters values were used as filters. Figure 20 shows that from a total number of possible bigrams of 72,818, filtering by document frequency (DF = 40) reduce this number to 735, and to 77 through the TV (TV = 0.98) filter. The significant decrease implied that the distribution of bigrams were highly sparse. Thus, although the mapping rate was 45.38% with 77 features (see Figure 21), the precision, recall and F-measure values reached zero (see Figure 22). By contrast, through document frequency filtering, the number of features was ten times higher than with term variance frequency (735 features) and the $F^L_{0.5}$ was kept at almost 70% (as shown in Figure 22). Without filtering, although the number of bigrams was 72,818, many noisy features were removed which decreased $F^L_{0.5}$ to 23.90%.

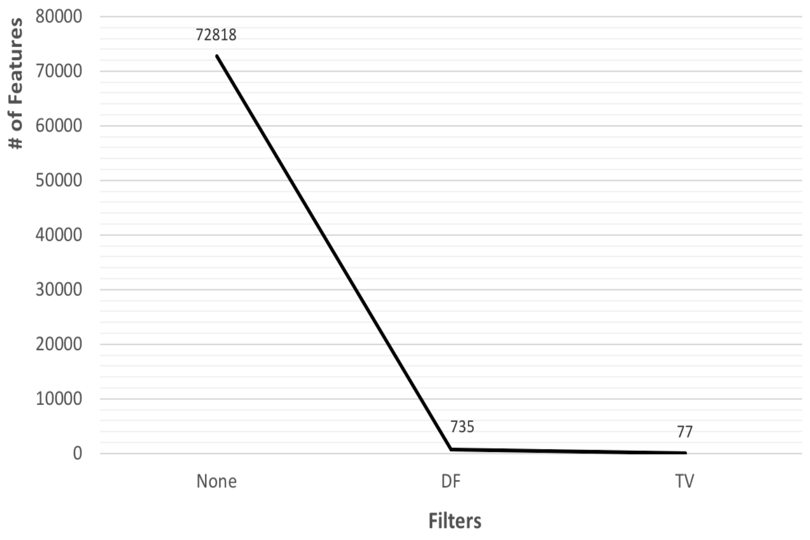


Figure 20: Number of Bigram Features

Due to the significant decrease in the number of bigrams, we can assume that trigrams will suffer even more from this sparse distribution. As shown in Tables 22 and 23 (in Appendix A), although the total number of trigrams was 106,084, all values of the mapping quality metrics were zero. This is because there are less common trigrams between snort alert messages and attack fields in CAPEC.

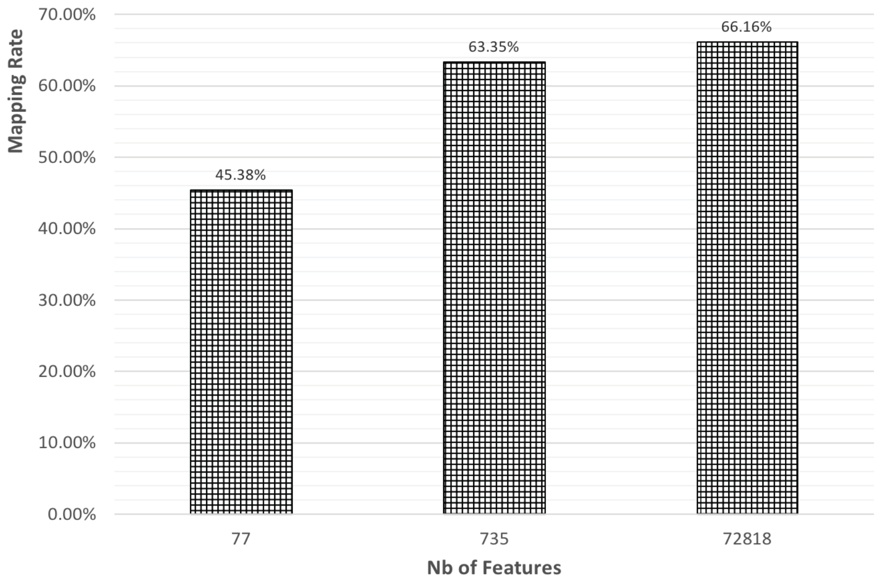


Figure 21: Mapping Rate of the Bigram Experiments

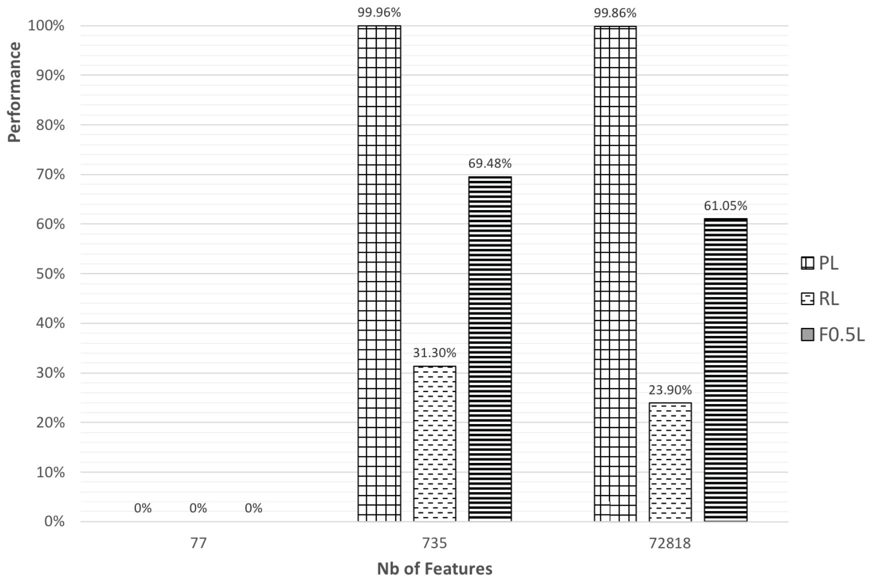


Figure 22: Results of the Bigram Experiments

Based on the analysis of the use of unigrams, bigrams and trigrams respectively, we continued conducting experiments on two kinds of n-grams mixtures: unigrams + bigrams and unigrams + bigrams + trigrams (which is equivalent to the baseline Charibdis system), in the hopes of identifying a better n-grams mixture contributing to the mapping quality.

4.1.3 Experiments Based on a Mixture of N-Grams

The purpose of this experiment is to identify if trigram can be beneficial to the mapping quality in a mixture of n-grams model. We first used the mixture of unigrams + bigrams to represent each snort alert message and attack field after tokenization, stemming and the removal of stop words. The algorithm pipeline is identical to the baseline Charibdis system. After we evaluated the unigrams + bigrams model, we replaced it with unigrams + bigrams + trigrams mixture model and followed the same pipeline to obtain the evaluation values. Table 11 shows the difference in the number of features between the two n-grams mixtures and Figure 23 illustrates the mapping quality of Experiment #3 and Experiment #7 (with DF = 40) in Tables 25 and 26. As the result shows, trigrams did not make difference in the baseline Charibdis system. Therefore, in the next experiments, only unigrams, bigrams and the mixture of unigrams + bigrams are used as features to represent documents.

Exp.	**Sim$_T$**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	40	0.98	Yes	209
Exp. #2	0	0	0.98	Yes	209
Exp. #3	0	40	0	Yes	1,573
Exp. #4	0	0	0	Yes	89,313
Exp. #5	0	40	0.98	Yes	260
Exp. #6	0	0	0.98	Yes	260
Exp. #7	0	40	0	Yes	2,104
Exp. #8	0	0	0	Yes	193,227

Table 11: Description of the Mixture N-Grams Experiments

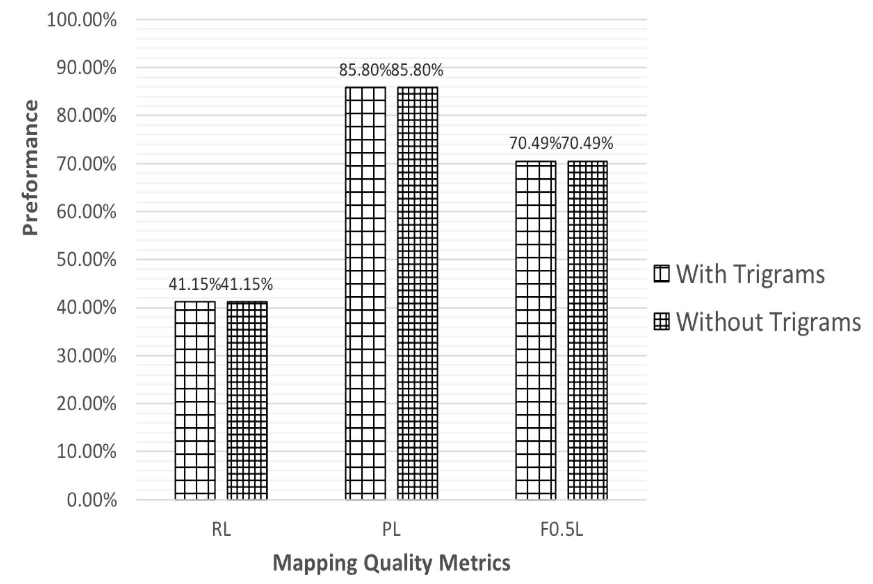


Figure 23: Results of Two N-Grams Mixtures

4.1.4 Overall Conclusion on N-Gram Feature Selection

As in the previous analysis (see Section 4.1.3), we found that unigrams are the most discriminating features contributing to the mapping quality, followed by bigrams; whereas trigrams do not contribute anything. Thus, we analysed the length distribution in snort messages and attack fields. Figure 24 shows the length distribution of snort messages and CAPEC fields for every 15 tokens. After the expansion of snort rule name descriptions, 22,817 messages were expanded to less than 15 tokens which accounts for 70% (22,817 / 32,246) of the total messages, while almost three quarters of attack fields contain more than 15 tokens. Figure 25 illustrates the number of correct and acceptable mappings with document frequency as 40 (DF = 40), snort messages and attack fields are represented by the mixture of unigrams + bigrams. Most of the mappings dropped in the $[30, 45)$ tokens zone followed by the $[15, 30)$ zone whereas only 97 mappings generated from the first zone. Thus, the short size of the snort alert messages is a vital factor which hinders features to yield good mapping result.

Overall, because of the small size of the snort alert messages, unigrams are the best choice to represent snort messages and CAPEC fields. Thus, we considered that if the snort alert messages can be supplemented, the number of overlapping bigrams could be increased and the lenient recall (R^L) and F-measure ($F^L_{0.5}$) could be improved. To

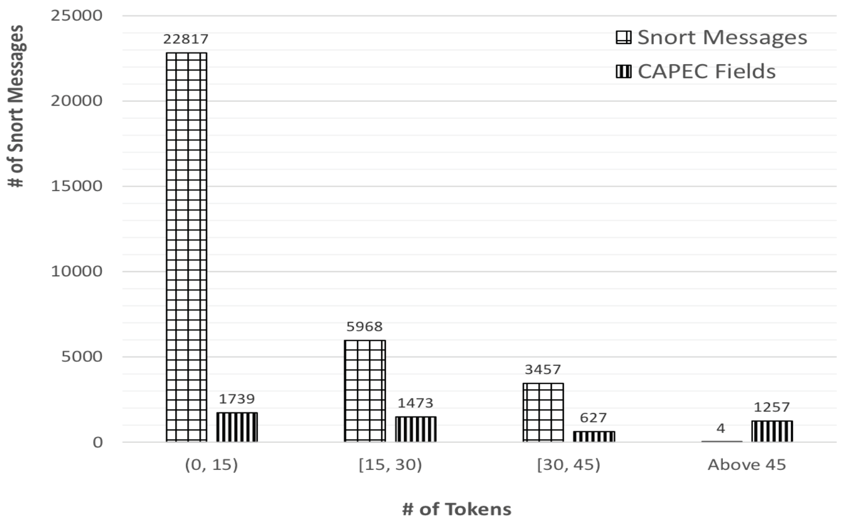


Figure 24: Distribution of the Length of Snort Messages and CAPEC Fields

supplement snort messages and try to decrease noisy features, we therefore used the related CVE entities specified at the end of each snort message (see Section 2.2.1.2) to extend snort messages. This is discussed in next section.

4.2 Snort Messages Expansion

To address the issue of the short snort alert messages, we investigated to use Common Vulnerabilities and Exposures (CVE) entities to supplement snort messages. Section 4.2.1 describes the pipeline of this snort expansion and Section 4.2.2 gives the analysis of the mapping results after the expansion.

4.2.1 Snort Messages Expansion Pipeline

As indicated in Section 2.2.1.2, the snort messages given to Charibdis include a reference to related CVE entity after the text of the snort message itself. For example, the input

---

26312;"PROTOCOL-SCADA WellinTech Kingview HMI history server buffer overflow attempt";"cve,2011-4536"

---

indicates that the message is related to the CVE entity `2011-4536`. Thus, the brief

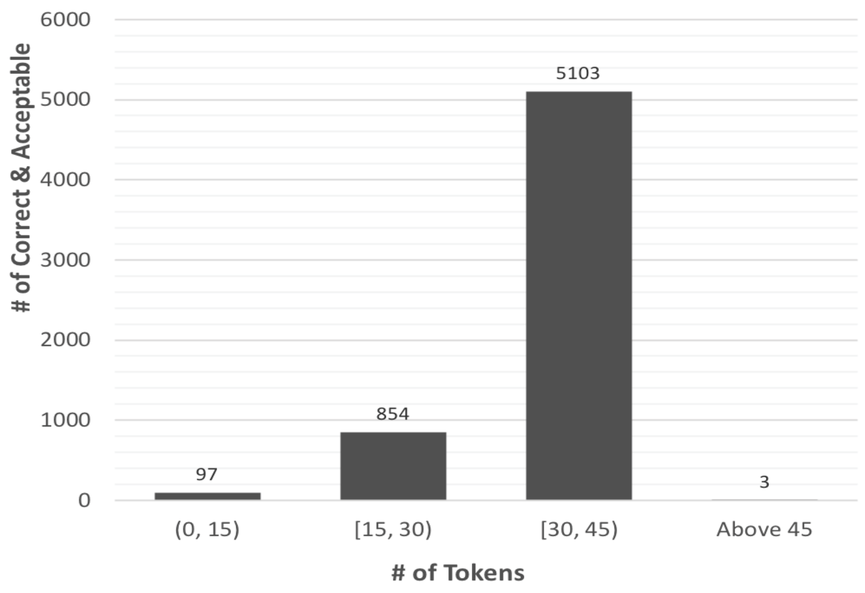


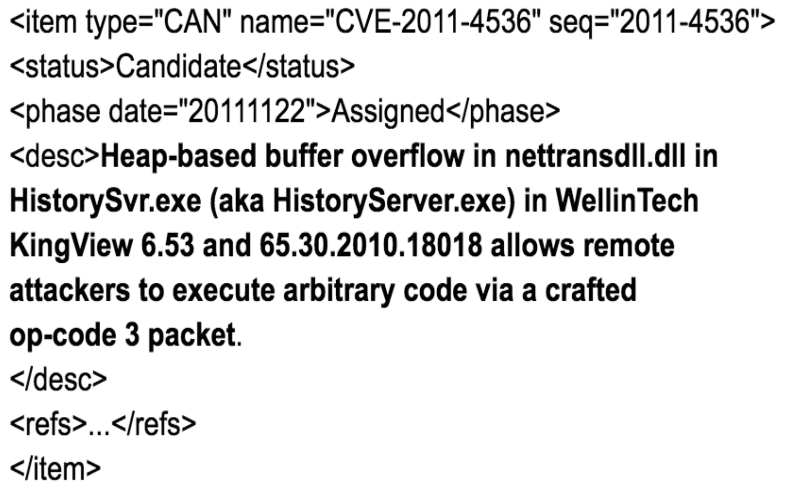
Figure 25: Number of Correct and Acceptable Mappings with the Mixture of Unigrams + Bigrams

description of the 2011-4536 CVE entity can be used to extend the snort message. As described in Section 2.1.1, CVE includes 98,375 entities in total. To search a specific entity from the large CVE dataset, we downloaded all of the CVE entities as an XML file. Figure 26 shows the `CVE-2011-4536` entity in the XML file. The text in bold indicates the natural language description of the entity.

The descriptions in each CVE entity contains 30 tokens on average which have been used to supplement snort messages through the pipeline indicated below:

1. The algorithm reads all of the CVE entities into memory, extracts CVE entity IDs as keys and the natural language descriptions as values. The (key: value) pairs are stored in a dictionary data structure in Python. The total size of the dictionary is 98,375, which is the number of CVE entities. Although the size is large, the time complexity of acquiring a CVE entity is $O(1)$ because the data structure is a hash. In our example in Figure 26, the (key: value) pair is `(2011-4536:  Heap-based buffer overflow ...  op-code 3 packet.)`

2. After the creation of the dictionary, the program parses the CVE entity ID in the input snort messages and use it as a key to get the associated descriptions in the dictionary. Then, the obtained CVE entity descriptions are concatenated at the end of the snort message to form a longer snort message. For instance,



```
<item type="CAN" name="CVE-2011-4536" seq="2011-4536">
<status>Candidate</status>
<phase date="20111122">Assigned</phase>
<desc>Heap-based buffer overflow in nettransdll.dll in
HistorySvr.exe (aka HistoryServer.exe) in WellinTech
KingView 6.53 and 65.30.2010.18018 allows remote
attackers to execute arbitrary code via a crafted
op-code 3 packet.
</desc>
<refs>...</refs>
</item>
```

Figure 26: Example of a CVE Entity

the extended snort message in our running example is:

```
PROTOCOL-SCADA WellinTech Kingview HMI history server buffer
overflow attempt Heap-based buffer overflow in nettransdll.dll
in HistorySvr.exe (aka HistoryServer.exe) in WellinTech KingView
6.53 and 65.30.2010.18018 allows remote attackers to execute
arbitrary code via a crafted op-code 3 packet.op-code 3 packet.
```

The average size of the expanded snort messages increased from 15 to 50. To analyse the increase in length of the snort messages after the CVE entity expansion, we measured the length distribution of new snort messages. Figure 27 shows a comparison of the length of messages with and without CVE entity expansion. As Figure 27 shows, the number of snort messages containing less than 15 tokens decreased by half after the CVE expansion, from 22,817 to 9,353, while a third of the snort messages (12,189 / 32,246) now contain more than 45 tokens.
After these two supplement steps, the new snort messages are also extended through snort rule name descriptions (see Section 3.1.1) and the algorithm of mapping sentences is identical as baseline system. Based on the extended snort messages, we again conducted unigrams, bigrams experiments respectively, as well as the mixture of unigrams + bigrams.

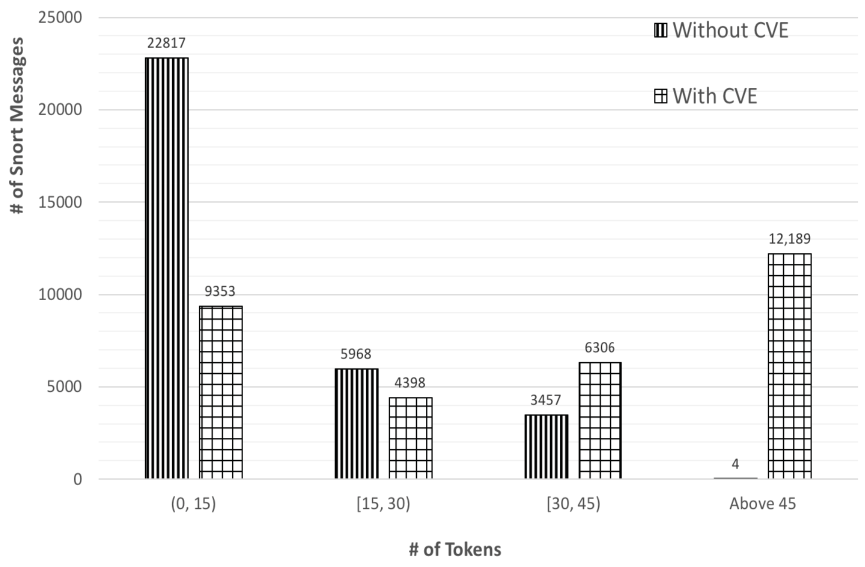


Figure 27: Comparison of the Length of Snort Messages with and without CVE Expansion

4.2.2 Results and Analysis

Using the expanded snort messages, we re-run the three successful experiments of Section 4.1: the use of unigrams only, the use of bigrams only and the mixture of unigrams + bigrams.

4.2.2.1 Experiment Based on Unigrams

Following the same mapping algorithm indicated in Section 2.2.1.3, only unigrams are used as features to represent extended snort alert messages and CAPEC attack fields. With the document frequency filter as 40 (DF = 40), Figure 28 shows that the values of R^L, P^L and $F^L_{0.5}$ all improved by 10% compared to the unigram performance in the baseline Charibdis system.

4.2.2.2 Experiment Based on Bigrams

In addition to the unigrams, bigrams also benefited from the use of CVE expansion. Figure 29 illustrates that the R^L increased from 31.30% in baseline system to 34.59%.

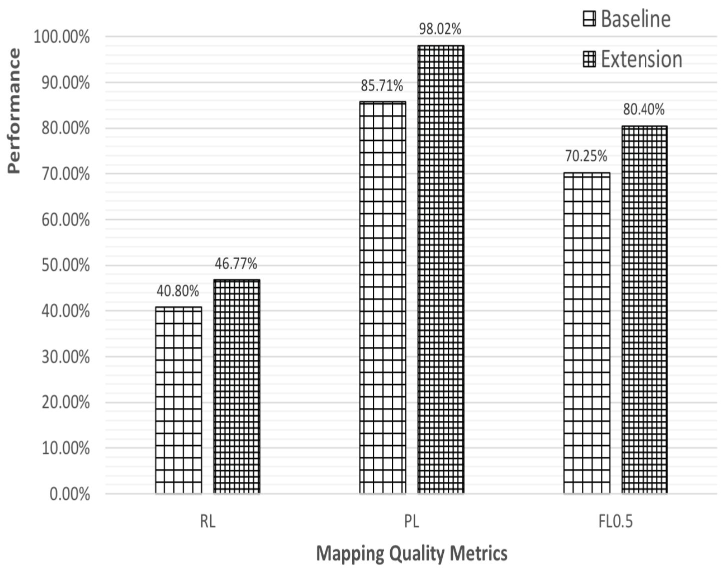


Figure 28: Comparison of the Mapping Quality on Unigrams with and without CVE Expansion

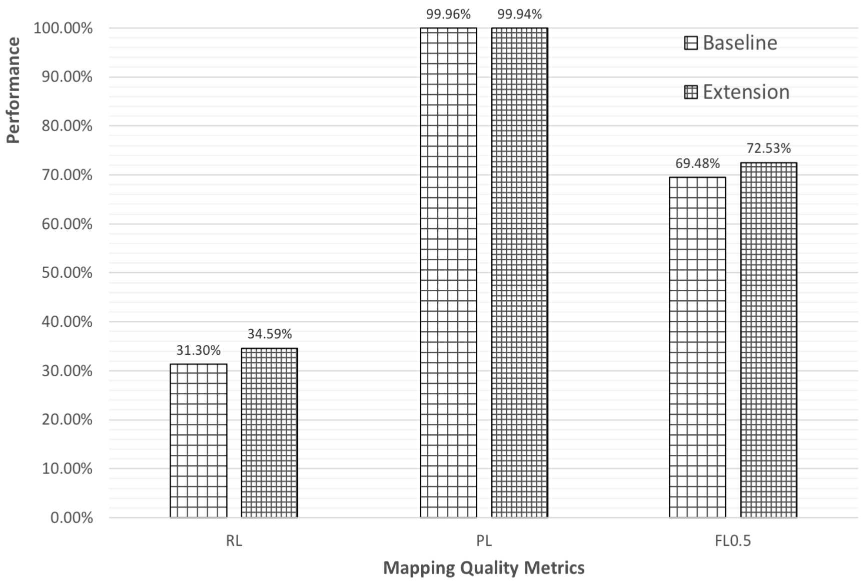


Figure 29: Comparison of the Mapping Quality on Bigrams with and without CVE Expansion

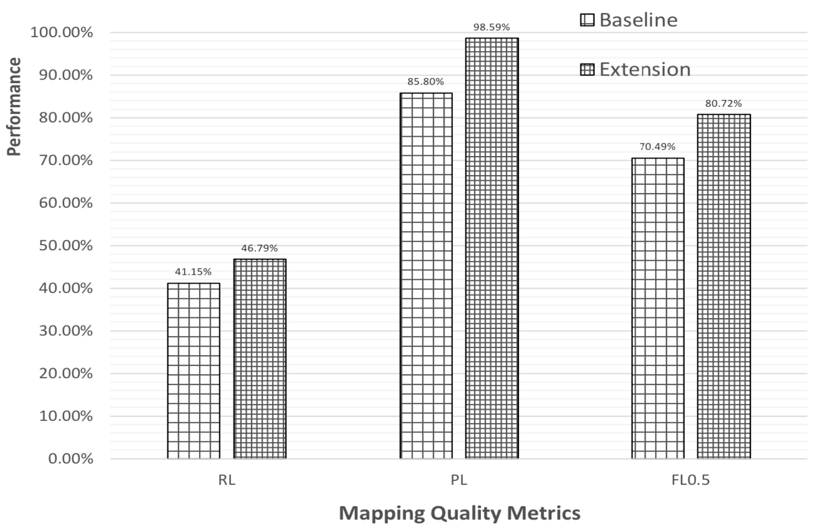


Figure 30: Comparison of the Mapping Quality of Mixture of Unigrams + Bigrams with and without CVE Expansion

4.2.2.3 Experiment Based on the N-Gram Mixture

Finally, the mixture of unigrams + bigrams to represent messages and fields also benefited from the CVE expansion. Figure 30 shows an improvement of P^L (13%) and a 10% increase in $F^L_{0.5}$.

4.2.3 Analysis of the CVE Expansion

The CVE entity expansion increases the performance of the baseline Charibdis system both in lenient precision and lenient recall using all feature combinations that we experimented with. However, two thirds of the snort messages distributed at the two sides in Figure 27, because only 47% (15,444 / 32,246) snort messages have the related CVE entity descriptions. For example, the message

7063;"EXPLOIT-KIT Multiple exploit kit jar file retrieved on non-standard port";

only has the snort ID and message content. Thus, the polarized length distribution of the extended snort messages limited the improvement of mapping quality.

In this chapter, we analyzed the contribution of the unigrams, bigrams, trigrams and two mixture feature models. Because of the small size of snort messages, unigrams seemed to contribute the most of the mapping quality. To address the short size issue, we used CVE entity descriptions to supplement snort messages and the performance of the system improved significantly as the $F_{0.5}^{L}$ increased from 70.49% to 80.72% with the mixture of unigrams and bigrams. However, the polarized length distribution of the extended snort messages seemed to prevent the increase of the mapping quality (see Figure 27) even further because only 47% of the snort messages have a related CVE entity. In order to supplement all of the messages, we investigated to use similar messages pre-clustering techniques before mapping. In the next chapter, we will describe and analyze the use of clustering.

Chapter 5

Pre-Clustering

In this chapter, we describe our second approach to improve the system performance: pre-clustering snort alert messages. After analyzing snort messages and discussed with our two security analysts, we noticed that many snort messages share similar content, hence it would seem natural that they be mapped to similar CAPEC fields. For example, these three example snort messages:

FILE-PDF Adobe Acrobat Reader embedded TTF bytecode memory corruption attempt
FILE-PDF Adobe Acrobat Reader TTF parsing bad cmap format attempt
FILE-PDF Adobe Reader embedded TTF interger overflow attempt

---

describe vulnerability exploits about *Adobe reader TTF* and should be mapped to similar CAPEC fields. To ensure this, we experiment with clustering the snort messages prior to mapping them. Each snort message within a cluster is then mapped to the same CAPEC field. Section 5.1 describes the K-Means clustering algorithm that we used and the pre-clustering pipeline used in the enhanced system. Section 5.2 analyses the mapping quality after clustering similar snort alert messages.

5.1 K-Means Clustering

In this section, we first describe the basics of K-Means clustering in Section 5.1.1, followed by a description of the pipeline using K-Means in the enhanced system in Section 5.1.2.

5.1.1 Introduction to K-Means

K-Means [Jain and Dubes, 1988] is a widely used unsupervised clustering algorithm. The goal of this algorithm is to group data into k clusters based on the similarity of the data and refine the clusters iteratively until producing the final results. The algorithm consists of four main steps:

1. The algorithm inputs are a collection of data and the number of described clusters k.

2. K cluster centroids are generated randomly from the dataset and each centroid represents a unique cluster.

3. Each data is assigned to its nearest centroid measured by the standard (L2) Euclidean distance (see Section 2.3.1.2). Data which share the same centroid are assigned to the same cluster. More specifically, the nearest centroid is defined as:
$$\arg\min_{c_i \in C} dist(c_i,\ x)^2$$
where $dist()$ is the the standard (L2) Euclidean distance, C is the collection of centroids, c_i is a centroid in the collection and x represents one data from the dataset.

4. The data in the same cluster are used to update the centroid of their cluster. Suppose that the data in the i^{th} cluster is the set S_i, then the update process of the centroid c_i is defined as:
$$c_i \;=\; \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

The algorithm repeats step 3 and 4 until meeting a stopping criteria, such as the limit on the iteration steps or the minimum distance threshold between data and centroid. In our system, we use the K-Means algorithm as implemented in the scikit-learn library and the stopping criteria was set to a maximum of 300 iterations.

5.1.2 Algorithm Pipeline

To try to increase the mapping quality and address the issue of the small size of snort messages, we attempted to pre-cluster snort message before mapping them.

Specifically, snort messages are first expanded by the snort rule name description (see Section 3.1.1), and the expanded snort alert messages were represented by unigrams after the removal of the punctuation and stop words. These unigrams are used in the document-by-matrix representation and each entry is the TF.IDF value. Messages which share similar content are clustered in the same cluster via the K-Means algorithm. Snort messages in the same cluster are then concatenated into a single long message, and the resulting longer messages is then mapped to CAPEC fields using the same mapping algorithm as the baseline system (see Section 2.2.1.3).

We experimented with various numbers of clusters (n) which as a side-effect also varied the length of the resulting message to map. The trade-off is that a larger number of clusters (n) should lead to a greater number of possible CAPEC fields being mapped to each snort messages, but should also lead to a shorter message and sparser representation. Figure 31 shows the average length of the new longer snort messages when using various number numbers of clusters. As shown in Figure 31, the length of snort messages varies from 23,539 (with 20 clusters) to 94 (with 5,000 clusters) and the length of the message is 235 (with 2,000 clusters) which is almost equivalent to the length of CAPEC fields (see Section 2.1.3). Based on these n clusters, we conducted the experiments and analyse the mapping quality in the next section.

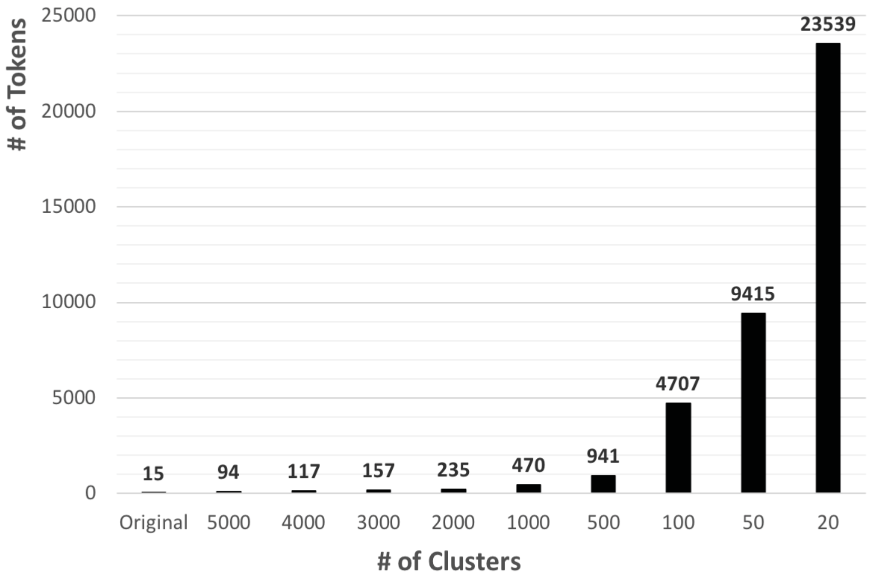


Figure 31: Distribution of Average Snort Length with Different Numbers of Clusters

5.2 Analysis of Pre-clustering

After clustering snort messages into n clusters using the K-Means algorithm (see Section 5.1) and concatenating longer messages, we used document frequency (DF) only and term variance frequency (TV) only as the filters similarly to the three experiments of Section 4.1: using unigrams only, using bigrams only and using the mixture of unigrams + bigrams to represent snort messages and CAPEC attack fields.

5.2.1 Experiments Based on Unigrams

Tables 12 and 13 show the number of features with document frequency (DF = 40) only and term variance frequency only (TV = 0.98) of each cluster based on the unigrams. These experiments gave raise to a feature set F where size ranges from 485 to 650 with the DF filter and from 650 to 1,218 with TV filter. Both the system with DF filter and that with TV filter perform best in the condition of 20 clusters (see Tables 38 and 39 in the Appendix). Figure 32 shows that the best $F^L_{0.5}$ achieved 80.50% when using the term variance frequency filter whereas this figure reached 71.68% with document frequency. Compared to the DF system, TV system performs better. In addition, Figure 33 shows the values of P^L, R^L and $F^L_{0.5}$ all increased by 10% through the use of the pre-clustering technique with term variance filter compared to the baseline system.

Exp.	**Sim$_\mathbf{T}$**	$\boldsymbol{n}$	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	20	40	0	Yes	485
Exp. #2	0	50	40	0	Yes	501
Exp. #3	0	100	40	0	Yes	522
Exp. #4	0	500	40	0	Yes	581
Exp. #5	0	1,000	40	0	Yes	591
Exp. #6	0	2,000	40	0	Yes	617
Exp. #7	0	3,000	40	0	Yes	631
Exp. #8	0	4,000	40	0	Yes	635
Exp. #9	0	5,000	40	0	Yes	650

Table 12: Description of the Pre-clustering on Unigrams Experiments only with Document Frequency

Exp.	Sim_T	n	DF	TV	Expansion	Nb of Features
Exp. #1	0	20	0	0.98	Yes	1,218
Exp. #2	0	50	0	0.98	Yes	1,125
Exp. #3	0	100	0	0.98	Yes	1,078
Exp. #4	0	500	0	0.98	Yes	1,015
Exp. #5	0	1,000	0	0.98	Yes	1,019
Exp. #6	0	2,000	0	0.98	Yes	951
Exp. #7	0	3,000	0	0.98	Yes	631
Exp. #8	0	4,000	0	0.98	Yes	635
Exp. #9	0	5,000	0	0.98	Yes	650

Table 13: Description of the Pre-clustering on Unigrams Experiments only with Term Variance Frequency

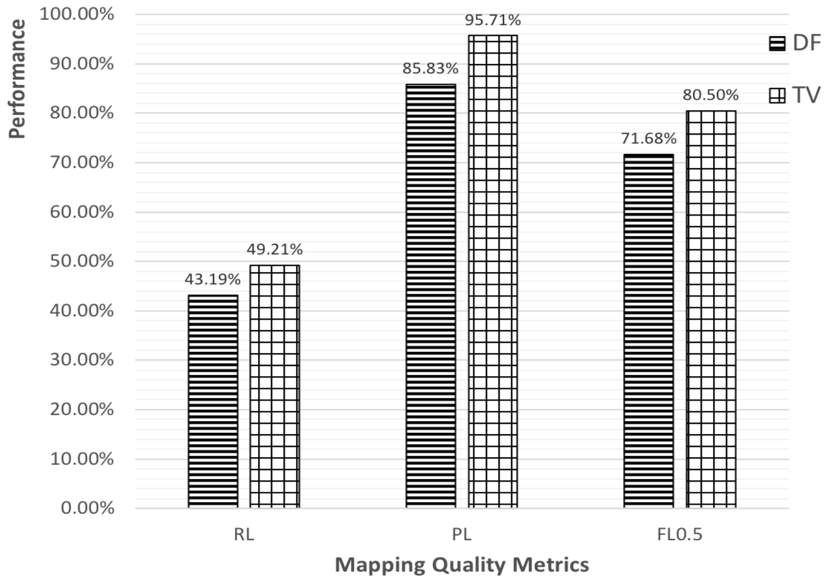


Figure 32: Comparison of Mapping Quality in Unigrams with DF only and TV only

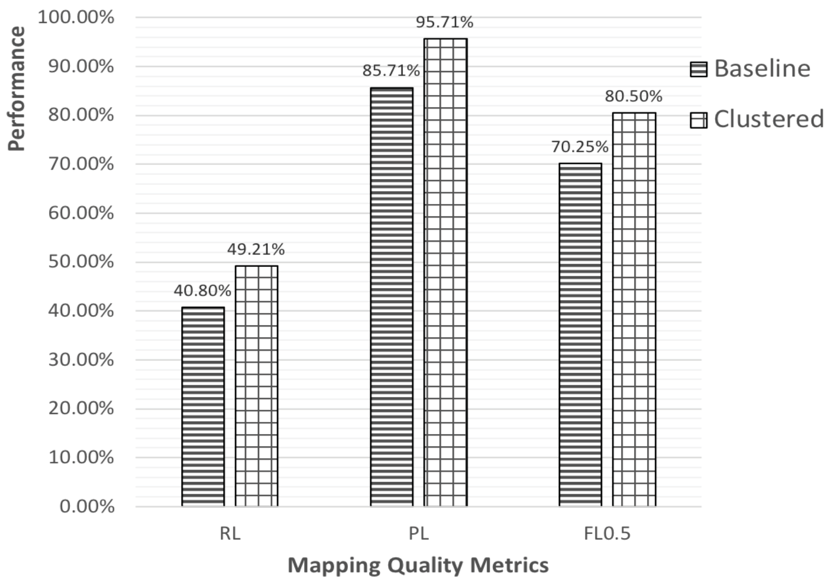


Figure 33: Comparison of Mapping Quality in Unigrams with 20 Clusters + TV and Baseline

5.2.2 Experiments Based on Bigrams

In addition to the unigram experiments, we used bigrams only to represent snort messages and attack fields. Using the document frequency as filter, the $F^L_{0.5}$ reached 73.79% with 3,000 clusters whereas the system with term variance frequency only achieved 60.27% in the same number of clusters (see Tables 42 and 45 in the Appendix). Compared to the bigram performance in the baseline system, Figure 34 illustrates that the pre-clustering technique improved the R^L from 31.30% to 36.40% without hurting the P^L (99.96%) and the $F^L_{0.5}$ also increased to 73.79%.

5.2.3 Experiments Based on Mixture of Unigrams + Bigrams

Finally, the mixture of unigrams + bigrams are used as features after clustering similar snort messages. The R^L reached 49.57% using the document frequency filter with 20 clusters and the P^L reached 87.42% which increased the mapping quality in baseline. Similarly, using term variance frequency as filter, the values of R^L and P^L achieved 48.79% and 97.11% respectively with 1,000 clusters and improved the $F_L^{0.5}$ to 80.99% which is the highest value so far. When comparing these two kinds of filtering methods, the term variance frequency (TV = 0.98) seems to perform the

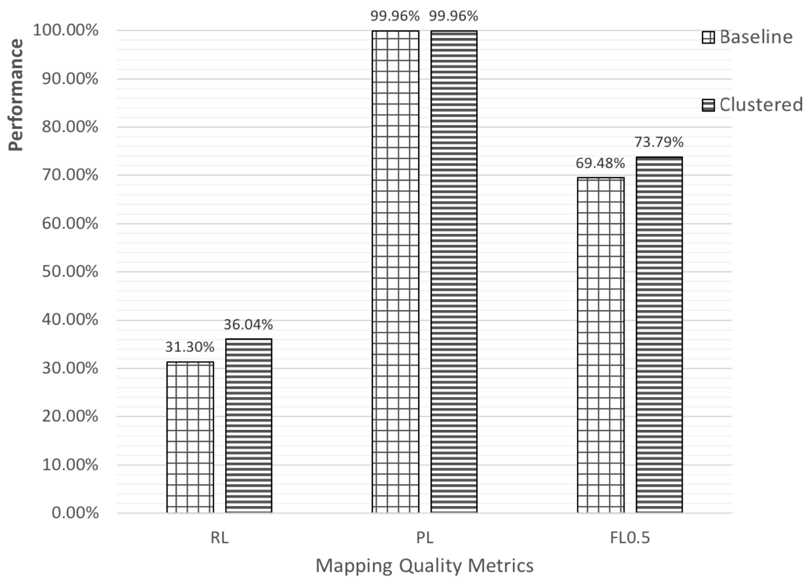


Figure 34: Comparison of Mapping Quality in Bigrams with 2000 Clusters + DF and Baseline

best in terms of both recall and precision. Figure 35 shows the comparison of the mapping quality with 1,000 clusters and the TV filtering and baseline in terms of all the values of lenient recall, precision and F-measure increased by almost 10%.

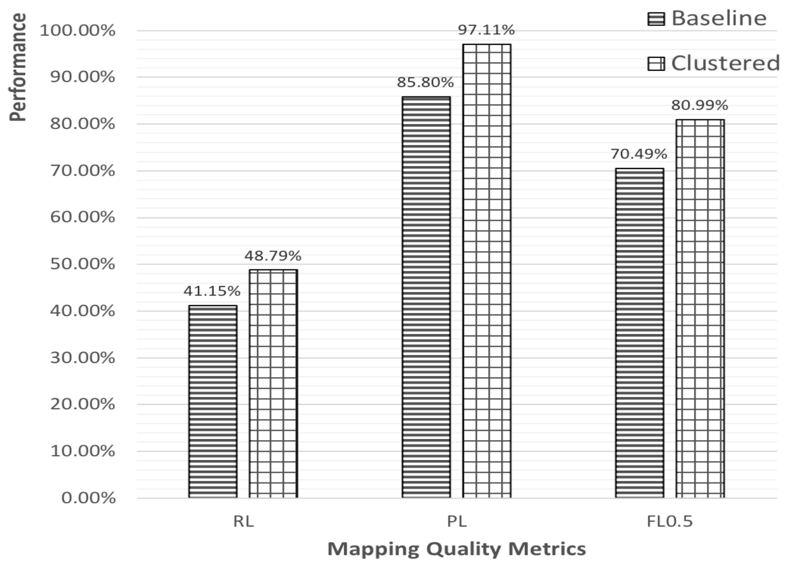


Figure 35: Comparison of Mapping Quality in Mixture of Unigrams + Bigrams with 1000 Clusters + TV and Baseline

5.2.4 Analysis of the Messages Pre-clustering

Based on the content similarity of snort messages, the pre-clustering method enhanced the mapping quality significantly compared to the baseline. Table 14 shows the results of the system with and without clustering as part of the pre-processing for various values of n when using unigrams + bigrams as features. As shown in Table 14, the best configurations in terms of F-measure are when using clustering with values of n between 500 and 3000 and using term variance frequency as filter. With these values, the results are not statistically different, with $F^L_{0.5} \approx 80\%$ and $F^L_1 \approx 65\%$. Recall itself has reached $\approx 48\%$ from a low 35.22% in the baseline (see Section 3.2.2). The best performance was achieved when $n = 1000$ and the F^L_1 was 64.84%. In addition, Table 14 shows the trade-off between the use of a smaller number of clusters (smaller n) which leads to a smaller number of possible output CAPEC fields and the use of a larger n which leads to a sparser snort representation. Hence leading to lower F-measures with $n \geq 3000$ and $n \leq 100$.

System	n	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	**Snort Length**
Baseline	n/a	97.96%	35.22%	72.23%	51.82%	15
Clustering	5000	86.62%	48.21%	74.71%	61.94%	94
Clustering	4000	86.41%	47.71%	74.35%	61.47%	117
Clustering	**3000**	97.11%	48.64%	80.97%	64.81%	157
Clustering	**2000**	96.82%	48.34%	80.65%	64.49%	235
Clustering	**1000**	**97.11%**	**48.67%**	**80.99%**	**64.84%**	470
Clustering	**500**	96.94%	48.51%	80.81%	64.67%	941
Clustering	100	96.00%	36.48%	72.38%	52.87%	4,707
Clustering	50	96.10%	36.58%	72.51%	52.99%	9,415
Clustering	20	96.13%	36.45%	72.42%	52.86%	23,539

Table 14: Results with the Mixture of Unigrams + Bigrams with Different Cluster Numbers + TV Filter

Compared to the Common Vulnerabilities and Exposures (CVE) expansion (see Section 4.2), the pre-clustering technique in the pre-processing step also performs better. Figure 36 shows how the pre-clustering contributes to the performance in comparison of CVE expansion (see Section 4.2) and the baseline alone, in terms of lenient recall (R^L) and F-measure (F_1^L). As shown in Figure 36, the R^L increased to 48.79% from 46.79% with the CVE expansion and the F_1^L improved to 64.84%. Both pre-clustering and CVE expansion improved the mapping quality significantly compared to the baseline.

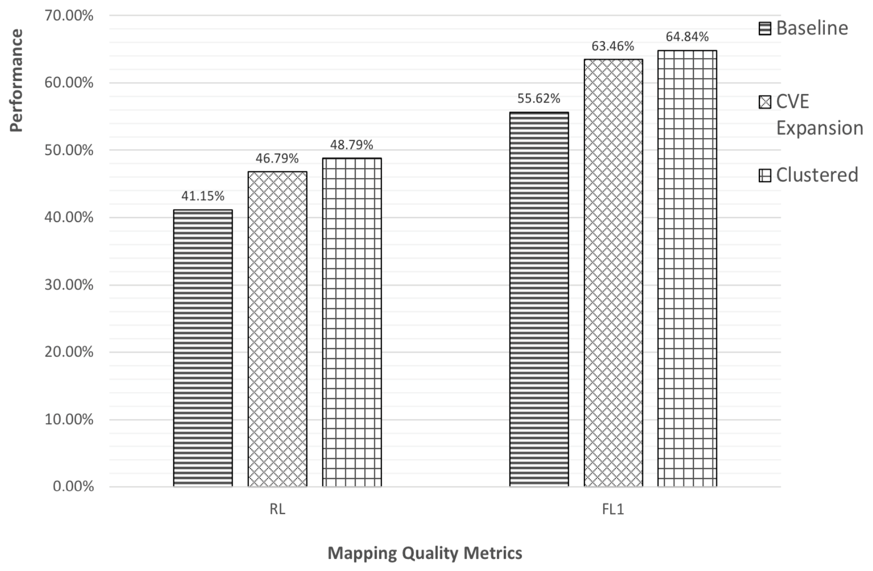


Figure 36: Analysis of the Mapping Quality Between the Baseline, the CVE Expansion and the Pre-clustering

In this chapter, we introduced the K-Means clustering algorithm and the pre-clustering algorithm pipeline used to enhance the mapping quality of the baseline system. We experimented with unigrams only, bigrams only and the mixture of unigrams + bigrams as features and analyzed the contribution of the pre-clustering technique. The mixture of unigrams + bigrams contributed most to the mapping quality. Using document frequency as the only filter and 20 clusters, the n-grams mixture achieved the highest lenient recall (R^L) at 49.57%. Likewise, using the variance term frequency filter only, the mixture increased all the values of R^L, P^L and F_1^L by 10%. However, when using a larger number of clusters ($n \geq 3000$), the sparse distribution of the snort messages seems to prevent an increase in the mapping quality. In order to tackle the sparse distribution problem, we investigated to use Latent Semantic Analysis (LSA) to decrease the dimension of feature set before mapping. In the next chapter, we will describe and analyze the use of LSA.

Chapter 6

Latent Semantic Analysis Approach

In this chapter, we describe the third approach to improve the mapping quality of the baseline system: Semantic Mapping by Latent Semantic Analysis. Recall from Section 2.3.4.1, that Latent Semantic Analysis (LSA) is widely used to reduce the feature size through removing unimportant latent topics. Section 6.1 introduces the basic idea of LSA and the algorithm pipeline in our system. Then, Section 6.2 analyses the enhanced mapping quality.

6.1 Latent Semantic Analysis

Mathematically, LSA decreases the dimension of the feature space to n topics using singular value decomposition (SVD) which is a well known matrix decomposition method. Any rectangular matrix can be decomposed into the product of three matrices:

$$C = U\Sigma V^T$$

where C is the rectangular matrix, U and V^T are two orthogonal matrices and Σ is a square diagonal matrix. Each diagonal value in Σ represents the importance of a unique topic and those smaller values which describe unimportant topics are removed in order to reduce the dimension of the features. The remained n topics highlight the documents similarity. In the enhanced system, the document-by-term matrix which is converted by the snort messages and CAPEC fields (see Section 2.2.1.3), is truncated into n important topics (dimensions) using the `TruncatedSVD` method in the scikit-learn library. For instance, suppose we have two snort messages and two CAPEC fields as follow.

Snort Messages:

d_1: `BROWSER-FIREFOX Mozilla Firefox IDB use-after-free attempt`

d_2: `SQL 1 = 1 - possible sql injection attempt`

CAPEC Attack Fields:

d_3: `This category is related to the WASC Threat Classification 2.0 item SQL Injection`

d_4: `Use a browser to manually explore the website and analyze how it is constructed.  Many browser's plug-in are available to facilitate the analysis or automate the URL discover`

When using unigrams only as features, the converted document-by-term matrix is:

	attempt	browser	inject	sql	url	...	websit
d_1	1	1	0	0	0	...	0
d_2	1	0	1	2	0	...	0
d_3	0	0	1	1	0	...	0
d_4	0	2	0	0	1	...	1

where each entry in this matrix is the term frequency in each document. In total, there are 28 unique unigram features in these four documents and this feature matrix suffers from a sparse distribution. After the feature dimension reduced is from 28 to 4 by using the latent semantic analysis method (TruncatedSVD method in scikit-learn library), the document-by-topics matrix becomes:

	$topic_1$	$topic_2$	$topic_3$	$topic_4$
d_1	0.97	0.89	2.66	-0.36
d_2	0.10	2.11	-0.15	1.58
d_3	0.03	2.32	-0.96	-1.29
d_4	4.06	-0.28	-0.62	0.06

where each entry in the document-by-topics matrix is the sum of the contribution of each feature in the corresponding document to the topic. For example, `0.97` represents the sum of the contribution of all the features in document 1 ($\mathtt{d}_1$) to topic 1 ($\mathtt{topic}_1$). The negative values indicate that the document is strongly unrelated to the topic.

6.1.1 Algorithm Pipeline

In the enhanced system, after the tokenizaion, the removal of stop words and stemming, term frequency only and TF.IDF (see Section 2.2.1.3) only are used as term weights separately in the document-by-term matrix. Without the filtering of document frequency (DF) and term variance frequency (TV), the latent semantic analysis method truncates snort messages and attack fields to n latent topics, where n varies from 5,000 to 100. Then, the cosine measure calculates the similarity between snort messages and attack fields represented in the document-by-topics matrix, and the 3 most similar fields are chosen as mapping result. The trade-off is that when using a larger number of topics, the details in snort messages and attack fields are taken into account in the similarity mapping whereas a smaller number of topics removes details and results only the summarization of messages and fields.

6.2 Analysis of the Latent Semantic Analysis Approach

With term frequency only and TF.IDF only, we conducted the three experiments similarly to Section 4.1: using unigrams only, using bigrams only and using the mixture of unigrams + bigrams to represent snort messages and CAPEC attack fields.

6.2.1 Experiments Based on Unigrams

Table 15 shows that the total number of unigrams features is 15,963 without the document frequency and term variance filtering. With 2,000 topics (Exp. #4), the R^L achieved 47.82% and F_1^L reached 64.34% with term frequency as term weight whereas the value of R^L was only 24.33% and the F_1^L achieved almost 39.15% with TF.IDF weight (see Tables 53 and 55 in the Appendix). Figure 37 compares the best performance of the baseline system and LSA methods with unigrams only. The R^L improved from 41.15% to 47.28% and the F_1^L increased by almost 10%, which is similar to the enhancement of the CVE expansion and pre-clustering (see Sections 4.1.1 and 5.2.1).

Exp.	**Sim$_\mathbf{T}$**	n	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	100	0	0	Yes	15,963
Exp. #2	0	500	0	0	Yes	15,963
Exp. #3	0	1,000	0	0	Yes	15,963
Exp. #4	0	2,000	0	0	Yes	15,963
Exp. #5	0	3,000	0	0	Yes	15,963
Exp. #6	0	4,000	0	0	Yes	15,963
Exp. #7	0	5,000	0	0	Yes	15,963

Table 15: Description of the Unigram with Latent Semantic Analysis Experiments

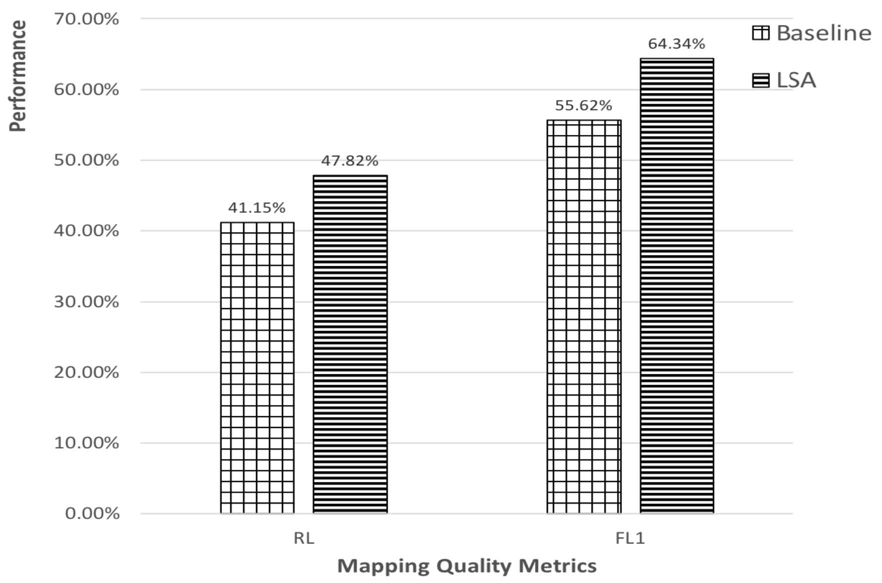


Figure 37: Comparison of the Mapping Quality of Unigrams Only Between Baseline and LSA

6.2.2 Experiments Based on Bigrams

With a smaller number of topics and bigrams only, the performance of the two weights (term frequency and TF.IDF) is almost equal, with $R^L \approx 35\%$ and $F_1^L \approx 52\%$ (see Tables 57 and 59 in the Appendix). Compared to the baseline system, the LSA method increased the mapping quality by 5%. Figure 38 shows that the R^L and the F_1^L improved to 35.69% and 52.06% respectively without hurting the value of lenient precision.

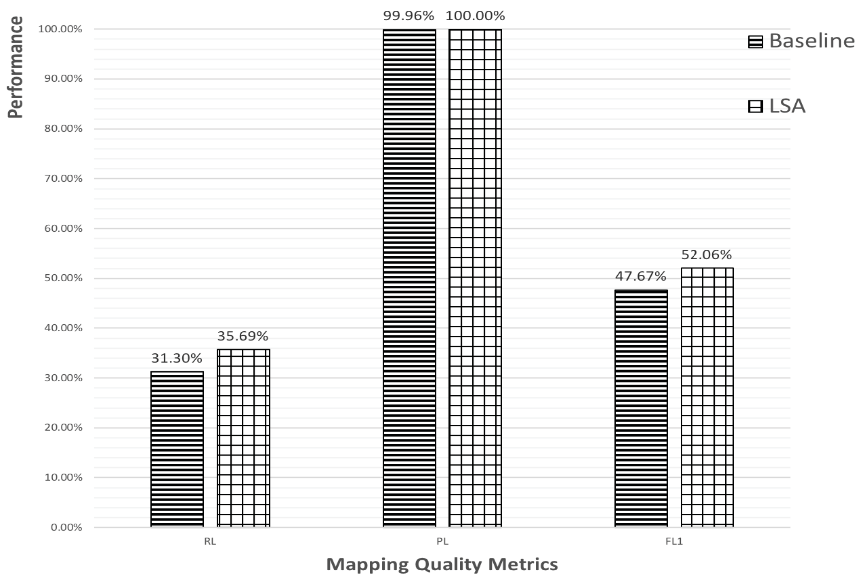


Figure 38: Comparison of the Mapping Quality of Bigrams Only Between Baseline and LSA

6.2.3 Experiments Based on Mixture of Unigrams + Bigrams

Similarly to the previous experiment, the mixture of unigrams + bigrams were used as features to represent snort messages and attack fields. Table 16 shows that the total number of unigrams + bigrams is 89,313 without document frequency and term variance frequency filtering. When reducing these almost 90,000 features to 5,000 and using term frequency as term weights, the R^L achieved 48% and the F_1^L reached 64.50%. Figure 39 shows the improvement that the latent semantic analysis method contributed. Compared to the baseline system, the LSA method not only increased the P^L to 98.30%, but the R^L also improved by 7% and the F_1^L achieved 64.50%.

6.2.4 Analysis of the Latent Semantic Analysis

Through removing unimportant topics, the latent semantic analysis method enhanced the performance of the baseline system. Table 17 shows the best mapping quality on

Exp.	**$\mathrm{Sim_T}$**	**n**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	100	0	0	Yes	89,313
Exp. #2	0	500	0	0	Yes	89,313
Exp. #3	0	1,000	0	0	Yes	89,313
Exp. #4	0	2,000	0	0	Yes	89,313
Exp. #5	0	3,000	0	0	Yes	89,313
Exp. #6	0	4,000	0	0	Yes	89,313
Exp. #7	0	5,000	0	0	Yes	89,313

Table 16: Description of the Mixture of Unigrams + Bigrams on Latent Semantic Analysis Experiments

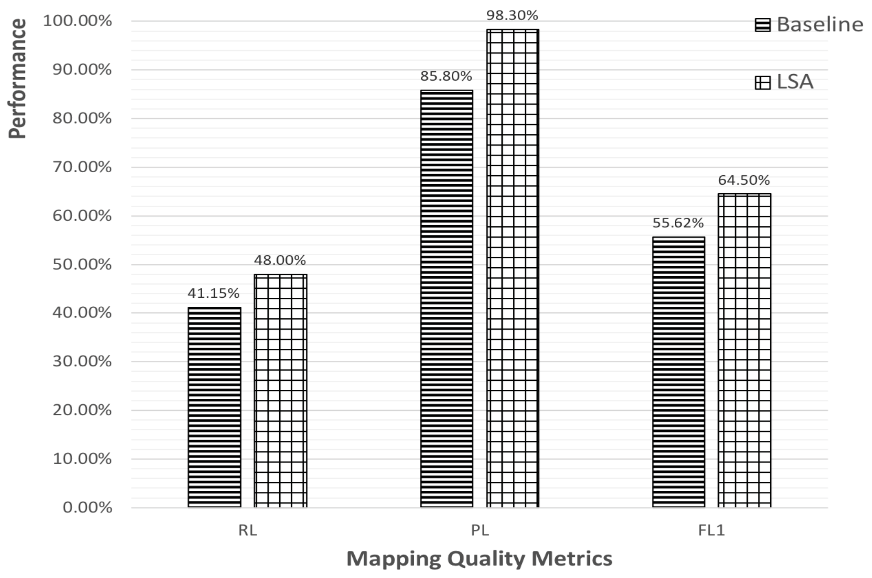


Figure 39: Comparison of the Mapping Quality of Mixture of Unigrams + Bigrams Between Baseline and LSA

the unigrams only, bigrams only and the mixture of unigrams + bigrams with the corresponding term weight and the number of topics. Compared to using TF.IDF as term weight, the term frequency is able to achieve a better mapping quality. As shown in Table 17, the best configurations in terms of F-measure are when using 5,000 as the number of topics, term frequency as term weight and the mixture of unigrams + bigrams as feature.

Feature	**Weight**	**Nb of Features**	$\boldsymbol{n}$	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{F_1^L}$
Unigrams	**TF**	**15,963**	**2,000**	**98.29%**	**47.82%**	**64.34%**
Unigrams	TF.IDF	15,963	4,000	99.97%	24.35%	39.17%
Bigrams	**TF**	**73,350**	**500**	**100.00%**	**35.69%**	**52.60%**
Bigrams	TF.IDF	73,350	100	100.00%	34.82%	51.65%
Mixture	**TF**	**89,313**	**5,000**	**98.30%**	**48.00%**	**64.50%**
Mixture	TF.IDF	89,313	3,000	100.00%	35.31%	52.19%

Table 17: Comparison of the Mapping Quality Between Term Frequency and TF.IDF

In this chapter, we have described the use of latent semantic analysis and the pipeline used to improve the mapping quality of the baseline system. Three experiments were conducted: unigrams only, bigrams only and the mixture of unigrams + bigrams with term frequency and TF.IDF respectively and the contribution of the LSA was analysed. With term frequency and the use of 5,000 topics, the mixture of unigrams + bigrams performs best with an F_1^L of 64.50%. In the next chapter, we will compare all of these three approaches and draw an overall conclusion.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis described several evaluation metrics and three approaches to enhance an existing Intrusion Detection System (IDS) for the automatic mapping of snort alert messages to known attack patterns. After evaluating the baseline system against a gold-standard, we found that unigrams are the most discriminating features contributing to the mapping quality, followed by bigrams; whereas trigrams do not contribute anything.

To address the short size issue highlighted in Section 1.1, we used Common Vulnerabilities and Exposures (CVE) entity descriptions to supplement snort messages and the performance of the system improved significantly as the F_1^L increased from 51.81% to 63.46% with the mixture of unigrams + bigrams. Compared to the CVE expansion, the pre-clustering technique proposed in Chapter 5, in the pre-processing step performs better. The n-gram mixture achieved the highest lenient recall (R^L) at 49.57% and the F_1^L increased to 48.67%. Also, the latent semantic analysis method of Chapter 6 enhanced the baseline system through reducing the feature size and the F_1^L improved to 48.00%

Finally, the recommended configurations are using the mixture of unigrams + bigrams as feature, clustering snort messages into 1,000 clusters and using term variance frequency as filter. With this configuration, the enhanced system performs best with the value of $R^L = 48.67\%$ and $F_1^L = 64.84\%$.

7.2 Future Work

As future work, it would be interesting to investigate the use of various automatic snort expansion methods. Currently, the snort rule name relies on hand-written term expansions and the detail snort messages are extended only through Common Vulnerabilities and Exposures (CVE) [MITRE, 2017b]. Because of the high workload restriction, hand written expansions cannot support snort rule name expansion when the number of snort messages is large. Also, the entities in CVE do not cover all snort messages, hence it cannot be used to supplement all snort messages. Thus, by using existing knowledge bases such as the Common Weakness Enumeration [MITRE, 2017c] or the Computer Security category in Wikipedia rather than relying on hand-written term expansion and Common Vulnerabilities and Exposures is necessary.

In addition, it would be interesting to look beyond the mapping of individual snort messages, and try to identify and match entire patterns/groups of snort messages as an indication of possible cyber attacks. In CAPEC, many attack patterns contain attack fields which have similar contents whereas these attack patterns are not similar. Thus, it could be an interesting research avenue to consider attack patterns as the smallest unit to map to snort messages.

Bibliography

[Allison and Dix, 1986] Allison, L. and Dix, T. I. (1986). A bit-string longest-common-subsequence algorithm. *Information Processing Letters*, 23(5):305–310.

[Ashoor and Gore, 2011] Ashoor, A. S. and Gore, S. (2011). Importance of intrusion detection system (IDS). *International Journal of Scientific and Engineering Research*, 2(1):1–4.

[Barrón-Cedeño and Rosso, 2009] Barrón-Cedeño, A. and Rosso, P. (2009). On automatic plagiarism detection based on n-grams comparison. In *European Conference on Information Retrieval*, pages 696–700. Springer.

[Bayuk, 2007] Bayuk, J. L. (2007). *Stepping Through the InfoSec Program*. Information Systems Audit and Control Association.

[Broder, 1997] Broder, A. Z. (1997). On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences 1997.*, pages 21–29. IEEE.

[Budanitsky and Hirst, 2006] Budanitsky, A. and Hirst, G. (2006). Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47.

[Chris et al., 1990] Chris, D. P. et al. (1990). Another stemmer. In *ACM Special Interest Group on Information Retrieval (SIGIR) Forum*, volume 24, pages 56–61.

[Clear, 1993] Clear, J. H. (1993). The digital word. chapter The British National Corpus, pages 163–187. MIT Press, Cambridge, MA, USA.

[Division, 2017] Division, N. C. S. (2017). National vulnerability database (NVD). `https://nvd.nist.gov`. Site Visited: March 15, 2018.

[Dobb's, 2007] Dobb's, D. (2007). SIEM: A market snapshot. `http://www.drdobbs.com/siem-a-market-snapshot/197002909`. Site Visited: March 1, 2018.

[Dolan et al., 2004] Dolan, B., Quirk, C., and Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 350–356. Association for Computational Linguistics.

[FIRST, 2018] FIRST (2018). Common vulnerability scoring system SIG. `https://www.first.org/cvss/`. Site Visited: March 1, 2018.

[Gomaa and Fahmy, 2013] Gomaa, W. H. and Fahmy, A. A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13).

[Gusfield, 1997] Gusfield, D. (1997). *Algorithms on strings, trees and sequences: Computer science and computational biology*. Cambridge University Press.

[Huang, 2008] Huang, A. (2008). Similarity measures for text document clustering. In *Proceedings of the Sixth New Zealand Computer Science Research Student Conference (NZCSRSC2008),* Christchurch, New Zealand, pages 49–56.

[Jaccard, 1901] Jaccard, P. (1901). Comparative study of floral distribution in a portion of the Alps and Jura. 37:547–579.

[Jain and Dubes, 1988] Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Jaro, 1989] Jaro, M. A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420.

[Jiang and Conrath, 1997] Jiang, J. J. and Conrath, D. W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*.

[Landauer and Dumais, 1997] Landauer, T. K. and Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104(2):211.

[Leacock and Chodorow, 1998] Leacock, C. and Chodorow, M. (1998). Combining local context and wordnet sense similarity for word sense identification. WordNet, an electronic lexical database. *The MIT Press*.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

[Lin et al., 1998] Lin, D. et al. (1998). An information-theoretic definition of similarity. In *International Conference on Machine Learning*, volume 98, pages 296–304.

[Loper and Bird, 2002] Loper, E. and Bird, S. (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70.

[Lyon et al., 2001] Lyon, C., Malcolm, J., and Dickerson, B. (2001). Detecting short passages of similar text in large document collections. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 118–125.

[Ma et al., 2018] Ma, X., Davoodi, E., Kosseim, L., and Scarabeo, N. (2018). Semantic mapping of security events to known attack patterns. In *International Conference on Applications of Natural Language to Information Systems*, pages 91–98. Springer.

[Manning and Schütze, 1999] Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*, volume 1:e25. Cambridge, MA, USA.

[Mihalcea et al., 2006] Mihalcea, R., Corley, C., Strapparava, C., et al. (2006). Corpus-based and knowledge-based measures of text semantic similarity. In *Association for the Advancement of Artificial Intelligence*, volume 6, pages 775–780.

[Miller, 1995] Miller, G. A. (1995). Wordnet: A lexical database for English. *Communication of the ACM*, 38(11):39–41.

[MITRE, 2017a] MITRE (2017a). Common attack pattern enumeration and classification (CAPEC) (2017). `https://capec.mitre.org/`. Site Visited: February 1, 2018.

[MITRE, 2017b] MITRE (2017b). Common vulnerabilities and exposures (CVE). `https://cve.mitre.org/`. Site Visited: March 1, 2018.

[MITRE, 2017c] MITRE (2017c). Common weakness enumeration (CWE) (2017). `https://cwe.mitre.org/index.html`. Site Visited: March 1, 2018.

[Monge and Elkan, 1997] Monge, A. and Elkan, C. (1997). An efficient domain-independent algorithm for detecting approximately duplicate database records. In *The Proceedings of the Special Interest Group on Management of Data (SIGMOD) 1997 Workshop on Data Mining and Knowledge Discovery*.

[Monge et al., 1996] Monge, A. E., Elkan, C., et al. (1996). The field matching problem: Algorithms and applications. In *Knowledge Discovery and Data Mining*, pages 267–270.

[Pedersen et al., 2004] Pedersen, T., Patwardhan, S., and Michelizzi, J. (2004). Wordnet:: Similarity: measuring the relatedness of concepts. In *Demonstration papers at HLT-NAACL 2004*, pages 38–41. Association for Computational Linguistics.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Porter, 1980] Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

[Porter, 2001] Porter, M. F. (2001). Snowball: A language for stemming algorithms. `http://snowball.tartarus.org/texts/introduction.html`. Site Visited: March 1, 2018.

[Resnik, 1995] Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*.

[Roesch et al., 1999] Roesch, M. et al. (1999). Snort: Lightweight intrusion detection for networks. In *Large Installation System Administration (LISA)*, volume 99, pages 229–238.

[Royer, 2005] Royer, C. (2005). Term representation with generalized latent semantic analysis. *Proceedings of Recent Advances in Natural Language Processing IV: selected papers from RANLP*, 292:45.

[Sahami and Heilman, 2006] Sahami, M. and Heilman, T. D. (2006). A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th International Conference on World Wide Web*, pages 377–386. ACM.

[Salton and McGill, 1986] Salton, G. and McGill, M. J. (1986). *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

[Scarabeo et al., 2015] Scarabeo, N., Fung, B. C., and Khokhar, R. H. (2015). Mining known attack patterns from security-related events. *PeerJ Computer Science*, 1:e25.

[Schatz et al., 2017] Schatz, D., Bashroush, R., and Wall, J. (2017). Towards a more representative definition of cyber security. *Journal of Digital Forensics, Security and Law*, 12(2):8.

[Schütze et al., 2008] Schütze, H., Manning, C. D., and Raghavan, P. (2008). *Introduction to information retrieval*, volume 39. Cambridge University Press.

[Smith and Waterman, 1981] Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Molecular Biology*, 147:195–197.

[NIST, 2017] NIST (2017). Information technology laboratory(ITL) (2017). `https://www.nist.gov/itl/about-itl`. Site Visited: March 1, 2018.

[Winkler, 1990] Winkler, W. E. (1990). String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage.

[Wu and Palmer, 1994] Wu, Z. and Palmer, M. (1994). Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics.

[ZDNet, 2006] ZDNet (2006). Novell buys e-security. `http://www.zdnet.com/article/novell-buys-e-security/`. Site Visited: March 1, 2018.

[Zipf, 2016] Zipf, G. K. (2016). *Human behavior and the principle of least effort: An introduction to human ecology*. Ravenio Books.

Appendix A

Details of Experimented Results

Unigrams on Baseline

Table 18 shows the lenient and strict measures when using unigrams only as the feature to represent snort alert messages and CAPEC attack fields.

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	**P^L**	**R^L**	**P^S**	**R^S**
Exp. #1	98.95%	6	4,280	108	97.54%	**29.12%**	0.14%	0.07%
Exp. #2	98.95%	6	4,280	108	97.54%	**29.12%**	0.14%	0.07%
Exp. #3	**99.40%**	**1.746**	**4,259**	**1,001**	**85.71%**	**40.80%**	**24.92%**	**18.93%**
Exp. #4	99.42%	1,789	1,718	2	99.94%	**23.83%**	50.98%	19.40%
Exp. #5	98.18%	11	14	10	71.43%	**0.17%**	31.43%	0.12%
Exp. #6	98.18%	11	14	10	71.43%	**0.17%**	31.43%	0.12%
Exp. #7	99.40%	100	10	38	74.32%	**0.75%**	67.57%	1.08%
Exp. #8	99.42%	102	1,581	1	99.94%	**11.43%**	6.06%	1.11%

Table 18: Results of the Unigram Experiments

Bigrams on Baseline

Table 19 shows the number of bigrams representing the snort messages and attack fields when using different combinations of document frequency (DF) and term variance filter (TV). Tables 20 and 21 show the mapping quality of bigrams.

Exp.	**Sim_T**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	40	0.98	Yes	74
Exp. #2	0	0	0.98	Yes	74
Exp. #3	0	40	0	Yes	681
Exp. #4	0	0	0	Yes	73,350

Table 19: Description of the Bigram Experiments

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	**P^L**	**R^L**	**P^S**	**R^S**
Exp. #1	45.38%	0	0	2	0.00%	**0.00%**	0.00%	0.00%
Exp. #2	45.38%	0	0	2	0.00%	**0.00%**	0.00%	0.00%
Exp. #3	**63.35%**	**1,539**	**3,068**	**2**	**99.96%**	**31.30%**	**33.39%**	**16.69%**
Exp. #4	66.16%	1,783	1,734	5	99.86%	**23.90%**	50.62%	19.33%

Table 20: Results of the Bigram Experiments

Exp.	**Lenient**			**Strict**		
	$F^L_{0.5}$	**F^L_1**	**F^L_2**	**$F^S_{0.5}$**	**F^S_1**	**F^S_2**
Exp. #1	**0.00%**	0.00%	0.00%	0.00%	0.00%	0.00%
Exp. #2	**0.00%**	0.00%	0.00%	0.00%	0.00%	0.00%
Exp. #3	**69.48%**	**47.67%**	**36.29%**	**27.82%**	**22.25%**	**47.67%**
Exp. #4	**61.05%**	27.98%	28.18%	38.25%	27.98%	22.06%

Table 21: F-Measures of the Bigram Experiments

Trigrams on Baseline

Tables 22, 23 and 24 demonstrate the number of trigrams and the related mapping quality.

Exp.	**$\mathbf{Sim_T}$**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	40	0.98	Yes	51
Exp. #2	0	0	0.98	Yes	51
Exp. #3	0	40	0	Yes	531
Exp. #4	0	0	0	Yes	103,914

Table 22: Description of the Trigram Experiments

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	0%	0	0	0	0.00%	0.00%	0.00%	0.00%
Exp. #2	0%	0	0	0	0.00%	0.00%	0.00%	0.00%
Exp. #3	4.71%	0	0	0	0.00%	0.00%	0.00%	0.00%
Exp. #4	5.33%	0	0	1	0.00%	0.00%	0.00%	0.00%

Table 23: Results of the Trigram Experiments

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Exp. #2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Exp. #3	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Exp. #4	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Table 24: F-Measures of the Trigram Experiments

Mixture of N-Grams on Baseline

After measuring each n-gram, Tables 25 and 26 show the mapping results and F-measure values when using the mixture of unigrams and bigrams and trigrams.

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	98.95%	6	5,178	107	97.97%	**35.22%**	0.11%	0.06%
Exp. #2	98.95%	6	5,178	107	97.97%	**35.22%**	0.11%	0.06%
Exp. #3	**99.40%**	**1,781**	**4,276**	**1,002**	**85.80%**	**41.15%**	**25.22%**	**19.31%**
Exp. #4	99.42%	1,781	2,519	0	100.00%	**29.22%**	41.42%	19.31%
Exp. #5	98.95%	6	5,178	108	97.96%	**35.22%**	0.11%	0.07%
Exp. #6	98.95%	6	5,178	108	97.96%	**35.22%**	0.11%	0.07%
Exp. #7	**99.40%**	**1,781**	**4,276**	**1,004**	**85.78%**	**41.15%**	**25.22%**	**19.31%**
Exp. #8	99.43%	1,744	2,516	2	99.95%	**28.94%**	40.92%	18.91%

Table 25: Results of the Mixture N-Grams Experiments

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**72.23%**	51.81%	40.40%	0.10%	0.08%	0.07%
Exp. #2	**72.23%**	51.81%	40.40%	0.10%	0.08%	0.07%
Exp. #3	**70.49%**	**55.62%**	**45.93%**	**23.77%**	**21.88%**	**20.26%**
Exp. #4	**67.36%**	45.22%	34.03%	33.70%	26.34%	21.62%
Exp. #5	**72.23%**	51.81%	40.40%	0.10%	0.08%	0.07%
Exp. #6	**72.23%**	51.81%	40.40%	0.10%	0.08%	0.07%
Exp. #7	**70.49%**	**55.62%**	**45.93%**	**23.77%**	**21.88%**	**20.26%**
Exp #8	**67.36%**	45.22%	34.03%	33.70%	26.34%	21.62%

Table 26: F-Measures of the Mixture N-Grams Experiments

Unigrams on Common Vulnerabilities and Exposures Extension

Tables 27 and 28 show the number of unigrams and the mapping results, in terms of lenient and strict values. Table 29 indicates the F-measure after the CVE extension.

Exp.	**Sim_T**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	40	0.98	Yes	337
Exp. #2	0	0	0.98	Yes	341
Exp. #3	0	40	0	Yes	1,383
Exp. #4	0	0	0	Yes	22,221

Table 27: Description of the Unigrams on CVE Extension Experiments

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	**P^L**	**R^L**	**P^S**	**R^S**
Exp. #1	99.16%	1,624	4,312	176	97.12%	40.33%	26.57%	17.61%
Exp. #2	99.16%	1,624	4,312	176	97.12%	40.33%	26.57%	17.61%
Exp. #3	**99.45%**	**1,735**	**5,149**	**139**	**98.02%**	**46.77%**	**24.70%**	**18.81%**
Exp. #4	99.47%	1,743	1,738	1	99.97%	23.65%	50.05%	18.90%

Table 28: Results of the Unigrams on CVE Extension Experiments

Exp.	**Lenient**			**Strict**		
	$F^L_{0.5}$	**F^L_1**	**F^L_2**	**$F^S_{0.5}$**	**F^S_1**	**F^S_2**
Exp. #1	**75.78%**	56.99%	45.67%	24.11%	21.18%	18.88%
Exp. #2	**75.78%**	56.99%	45.67%	24.11%	21.18%	18.88%
Exp. #3	**80.40%**	**63.32%**	**52.23%**	**23.24%**	**21.36%**	**19.75%**
Exp. #4	**60.75%**	38.25%	27.91%	37.64%	27.44%	21.58%

Table 29: F-Measures of the Unigrams on CVE Extension Experiments

Bigrams on Common Vulnerabilities and Exposures Extension

Tables 30, 31 and 32 show the mapping quality and F-measure when using bigrams to represent CAPEC attack fields and extended snort messages.

Exp.	**Sim_T**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	40	0.98	Yes	217
Exp. #2	0	0	0.98	Yes	222
Exp. #3	0	40	0	Yes	2,312
Exp. #4	0	0	0	Yes	113,180

Table 30: Description of the Bigrams on CVE Extension Experiments

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	**P^L**	**R^L**	**P^S**	**R^S**
Exp. #1	76.94%	0	0	0	0.00%	0.00%	0.00%	0.00%
Exp. #2	76.94%	0	0	0	0.00%	0.00%	0.00%	0.00%
Exp. #3	**83.88%**	**1,708**	**3,384**	**3**	**99.94%**	**34.59%**	**33.52%**	**18.52%**
Exp. #4	85.31%	1,712	1,693	4	99.88%	23.13%	50.22%	18.56%

Table 31: Results of the Bigrams on CVE Extension Experiments

Exp.	**Lenient**			**Strict**		
	$F^L_{0.5}$	**F^L_1**	**F^L_2**	**$F^S_{0.5}$**	**F^S_1**	**F^S_2**
Exp. #1	**0.00%**	0.00%	0.00%	0.00%	0.00%	0.00%
Exp. #2	**0.00%**	0.00%	0.00%	0.00%	0.00%	0.00%
Exp. #3	**72.53%**	**51.40%**	**39.80%**	**28.84%**	**23.85%**	**20.34%**
Exp. #4	**60.04%**	37.56%	27.33%	37.44%	27.10%	21.24%

Table 32: F-Measures of the Bigrams on CVE Extension Experiments

Mixture of Unigrams + Bigrams on Common Vulnerabilities and Exposures Expansion

Compared with using unigrams only and bigrams only, the mixture of unigrams and bigrams performs best.

Exp.	**Sim_T**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	40	0.98	Yes	554
Exp. #2	0	0	0.98	Yes	563
Exp. #3	0	40	0	Yes	3,695
Exp. #4	0	0	0	Yes	135,401

Table 33: Description of the Mixture of Unigrams + Bigrams on CVE Extension Experiments

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	**P^L**	**R^L**	**P^S**	**R^S**
Exp. #1	99.16%	1,693	5,145	174	97.51%	46.46%	24.14%	18.35%
Exp. #2	99.16%	1,693	5,145	174	97.51%	46.46%	24.14%	18.35%
Exp. #3	**99.45%**	**1,743**	**5,144**	**98**	**98.59%**	**46.79%**	**24.95%**	**18.90%**
Exp. #4	99.47%	1,732	2,524	1	99.97%	28.91%	40.68%	18.78%

Table 34: Results of the Mixture of Unigrams + Bigrams on CVE Extension Experiments

Exp.	**Lenient**			**Strict**		
	$F^L_{0.5}$	**F^L_1**	**F^L_2**	**$F^S_{0.5}$**	**F^S_1**	**F^S_2**
Exp. #1	**79.94%**	62.93%	51.89%	22.71%	20.85%	19.28%
Exp. #2	**79.94%**	62.93%	51.89%	22.71%	20.85%	19.28%
Exp. #3	**80.72%**	**63.46%**	**52.28%**	**23.45%**	**21.50%**	**19.86%**
Exp. #4	**67.03%**	44.85%	33.70%	32.99%	25.69%	21.04%

Table 35: F-Measures of the Mixture of Unigrams + Bigrams on CVE Extension Experiments

Unigrams on Pre-clustering Experiments

Using document frequency (DF) and term variance frequency (TV) separately and unigrams as features, Tables 36, 37, 38 and 39 show the mapping quality values and F-measures when clustering similar snort messages into one cluster.

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	**100.00%**	**1,860**	**4,498**	**1,049**	**85.83%**	**43.19%**	**25.11%**	**20.16%**
Exp. #2	**100.00%**	**1,859**	**4,424**	**1,249**	**83.41%**	**42.68%**	**24.68%**	**20.15%**
Exp. #3	100.00%	1,858	4,357	1,054	85.50%	**42.22%**	25.56%	20.14%
Exp. #4	100.00%	1,841	4,386	1,003	86.12%	**42.30%**	25.46%	19.96%
Exp. #5	100.00%	1,832	4,376	1,009	86.01%	**42.17%**	25.38%	19.86%
Exp. #6	100.00%	1,829	4,381	1,007	86.04%	**42.19%**	25.34%	19.83%
Exp. #7	100.00%	1,831	4,378	1,003	86.09%	**42.18%**	25.38%	19.85%
Exp. #8	100.00%	1,819	4,334	997	86.05%	**41.80%**	25.44%	19.72%
Exp. #9	100.00%	1,809	4,338	995	86.06%	**41.76%**	25.32%	19.61%

Table 36: Results of the Unigrams on Pre-clustering Experiments with Document Frequency

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**71.68%**	**57.47%**	**47.96%**	**23.93%**	**22.37%**	**20.99%**
Exp. #2	**70.05%**	**56.47%**	**47.30%**	**23.62%**	**22.19%**	**20.92%**
Exp. #3	**70.95%**	56.53%	46.98%	24.25%	22.53%	21.03%
Exp. #4	**71.34%**	56.74%	47.10%	24.13%	22.38%	20.86%
Exp. #5	**71.21%**	56.60%	46.96%	24.04%	22.28%	20.76%
Exp. #6	**71.23%**	56.62%	46.98%	24.00%	22.25%	20.73%
Exp. #7	**71.25%**	56.62%	46.97%	24.04%	22.28%	20.75%
Exp. #8	**71.02%**	56.27%	46.59%	24.04%	22.22%	20.65%
Exp. #9	**71.00%**	56.23%	46.55%	23.93%	22.10%	20.54%

Table 37: F-Measures of the Unigrams on Pre-clustering Experiments with Document Frequency

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	**100.00%**	**1,859**	**5,385**	**324**	**95.71%**	**49.21%**	**24.56%**	**20.15%**
Exp. #2	**100.00%**	**1,849**	**5,363**	**327**	**95.66%**	**49.00%**	**24.52%**	**20.05%**
Exp. #3	100.00%	1,845	5,296	337	95.49%	**48.51%**	24.67%	20.01%
Exp. #4	100.00%	1,841	5,281	317	95.74%	**48.39%**	24.75%	19.96%
Exp. #5	100.00%	1,827	5,281	317	95.73%	**48.29%**	24.61%	19.81%
Exp. #6	100.00%	96	5,268	337	94.08%	**36.44%**	1.68%	1.04%
Exp. #7	100.00%	94	5,269	308	94.56%	**36.43%**	1.65%	1.02%
Exp. #8	99.78%	49	4,339	1,198	78.55%	**29.81%**	0.87%	0.53%
Exp. #9	99.56%	50	4,329	1,190	78.63%	**29.75%**	0.89%	0.54%

Table 38: Results of the Unigrams on Pre-clustering Experiments with Term Variance Frequency

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**80.50%**	**65.00%**	**54.51%**	**23.53%**	**22.14%**	**20.90%**
Exp. #2	**80.35%**	**64.80%**	**54.30%**	**23.47%**	**22.06%**	**20.80%**
Exp. #3	**80.00%**	64.34%	53.81%	23.57%	22.09%	20.79%
Exp. #4	**80.06%**	64.28%	53.70%	23.61%	22.09%	20.76%
Exp. #5	**80.01%**	64.20%	53.60%	23.47%	21.94%	20.61%
Exp. #6	**71.47%**	52.53%	41.53%	1.49%	1.28%	1.12%
Exp. #7	**71.69%**	52.60%	41.54%	1.47%	1.26%	1.10%
Exp. #8	**59.19%**	43.22%	34.03%	0.77%	0.66%	0.57%
Exp. #9	**59.18%**	43.17%	33.97%	0.79%	0.67%	0.58%

Table 39: F-Measures of the Unigrams on Pre-clustering Experiments with Term Variance Frequency

Bigrams on Pre-clustering Experiments

Tables 40, 41, 42, 43, 44 and 45 show the results in using bigrams only as feature to represent snort messages and attack fields.

Exp.	**Sim_T**	n	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	20	40	0	Yes	87
Exp. #2	0	50	40	0	Yes	89
Exp. #3	0	100	40	0	Yes	91
Exp. #4	0	500	40	0	Yes	181
Exp. #5	0	1,000	40	0	Yes	218
Exp. #6	0	2,000	40	0	Yes	274
Exp. #7	0	3,000	40	0	Yes	309
Exp. #8	0	4,000	40	0	Yes	332
Exp. #9	0	5,000	40	0	Yes	348

Table 40: Description of the Bigrams on Pre-clustering Experiments with Document Frequency

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	83.07%	423	0	0	100.00%	**2.87%**	100.00%	4.58%
Exp. #2	78.48%	299	0	0	100.00%	**2.03%**	100.00%	3.24%
Exp. #3	78.13%	383	0	0	100.00%	**2.60%**	100.00%	4.15%
Exp. #4	76.26%	1,895	1,746	2	99.94%	**24.73%**	52.01%	20.54%
Exp. #5	**75.56%**	**1,832**	**3,471**	**2**	**99.96%**	**36.03%**	**34.53%**	**19.86%**
Exp. #6	**74.48%**	**1,809**	**3,475**	**2**	**99.96%**	**35.90%**	**34.22%**	**19.61%**
Exp. #7	**73,61%**	**1,828**	**3,477**	**2**	**99.96%**	**36.04%**	**34.44%**	**19.82%**
Exp. #8	72.27%	1,741	3,438	2	99.96%	**35.18%**	33.60%	18.87%
Exp. #9	71.56%	1,742	3,447	2	99.66%	**35.25%**	33.55%	18.88%

Table 41: Results of the Bigrams on Pre-clustering Experiments with Document Frequency

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**12.88%**	5.58%	3.56%	19.37%	8.77%	5.66%
Exp. #2	**9.39%**	3.98%	2.52%	14.35%	6.28%	4.02%
Exp. #3	**11.78%**	5.07%	3.23%	17.80%	7.97%	5.13%
Exp. #4	**62.15%**	39.66%	29.12%	39.82%	29.45%	23.37%
Exp. #5	**73.77%**	**52.96%**	**41.31%**	**30.09%**	**25.22%**	**21.71%**
Exp. #6	**73.67%**	**52.82%**	**41.17%**	**29.78%**	**24.93%**	**21.44%**
Exp. #7	**73.79%**	**52.98%**	**41.32%**	**30.01%**	**25.16%**	**21.66%**
Exp. #8	**73.06%**	52.05%	40.42%	29.06%	24.17%	20.69%
Exp. #9	**73.12%**	52.12%	40.49%	29.04%	24.17%	20.69%

Table 42: F-Measures of the Bigrams on Pre-clustering Experiments with Document Frequency

Exp.	$\mathbf{Sim_T}$	n	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	20	0	0.98	Yes	1,838
Exp. #2	0	50	0	0.98	Yes	1,721
Exp. #3	0	100	0	0.98	Yes	1,656
Exp. #4	0	500	0	0.98	Yes	1,528
Exp. #5	0	1,000	0	0.98	Yes	1,494
Exp. #6	0	2,000	0	0.98	Yes	1,313
Exp. #7	0	3,000	0	0.98	Yes	1,125
Exp. #8	0	4,000	0	0.98	Yes	993
Exp. #9	0	5,000	0	0.98	Yes	905

Table 43: Description of the Bigrams on Pre-clustering Experiments with Term Variance Frequency

Exp.	MR	MQ			Lenient		Strict	
		Correct	Acceptable	Incorrect	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	**93.91%**	**1,812**	**1,753**	**2**	**99.94%**	**24.22%**	**50.79%**	**19.64%**
Exp. #2	**88.10%**	**1,800**	**1,742**	**2**	**99.94%**	**24.06%**	**50.79%**	**19.51%**
Exp. #3	85.82%	1,797	1,737	2	99.94%	**24.01%**	50.82%	19.48%
Exp. #4	81.71%	1,741	1,731	2	99.94%	**23.59%**	50.11%	18.87%
Exp. #5	80.28%	1,708	1,702	2	99.94%	**23.16%**	50.05%	18.52%
Exp. #6	77.93%	1,700	1,694	2	99.94%	**23.06%**	50.05%	18.43%
Exp. #7	77,35%	1,716	1,712	2	99.94%	**23.29%**	50.03%	18.60%
Exp. #8	75.82%	1,685	1,682	2	99.94%	**22.87%**	50.01%	18.27%
Exp. #9	74.61%	1,129	1,129	2	99.91%	**15.34%**	49.95%	12.24%

Table 44: Results of the Bigrams on Pre-clustering Experiments with Term Variance Frequency

Exp.	Lenient			Strict		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**61.49%**	**38.99%**	**28.54%**	**38.56%**	**28.33%**	**22.39%**
Exp. #2	**61.29%**	**38.79%**	**28.37%**	**38.46%**	**28.19%**	**22.25%**
Exp. #3	**61.22%**	38.72%	28.31%	38.45%	28.17%	22.22%
Exp. #4	**60.66%**	38.17%	27.84%	37.65%	27.42%	21.56%
Exp. #5	**60.10%**	37.61%	27.37%	37.34%	27.03%	21.19%
Exp. #6	**59.96%**	37.47%	27.25%	37.27%	26.94%	21.10%
Exp. #7	**60.27%**	37.77%	27.51%	37.39%	27.12%	21.28%
Exp. #8	**59.71%**	37.23%	27.04%	37.11%	26.76%	20.92%
Exp. #9	**47.52%**	26.59%	18.46%	30.91%	19.66%	14.41%

Table 45: F-Measures of the Bigrams on Pre-clustering Experiments with Term Variance Frequency

Mixture of Unigrams + Bigrams on Pre-clustering Experiments

When using the mixture of unigrams and bigrams as feature, Tables 46, 47, 48, 49, 50 and 51 indicate the mapping results.

Exp.	**Sim_T**	**n**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	20	40	0	Yes	572
Exp. #2	0	50	40	0	Yes	590
Exp. #3	0	100	40	0	Yes	613
Exp. #4	0	500	40	0	Yes	763
Exp. #5	0	1,000	40	0	Yes	809
Exp. #6	0	2,000	40	0	Yes	891
Exp. #7	0	3,000	40	0	Yes	940
Exp. #8	0	4,000	40	0	Yes	967
Exp. #9	0	5,000	40	0	Yes	998

Table 46: Description of the Mixture of Unigrams + Bigrams on Pre-clustering Experiments with Document Frequency

Exp.	MR	MQ			Lenient		Strict	
		Correct	Acceptable	Incorrect	P^L	R^L	P^S	R^S
Exp. #1	**100.00%**	**1,859**	**5,437**	**1,049**	**87.42%**	**49.57%**	**22.27%**	**20.15%**
Exp. #2	**100.00%**	**1,858**	**5,363**	**1,249**	**85.25%**	**49.06%**	**21.93%**	**20.14%**
Exp. #3	100.00%	1,857	5,296	997	87.76%	**48.60%**	22.78%	20.13%
Exp. #4	100.00%	1,838	5,328	1,027	87.46%	**48.68%**	22.43%	19.93%
Exp. #5	100.00%	1,836	4,483	1,006	86.26%	**42.93%**	25.06%	19.90%
Exp. #6	99.99%	1,839	4,459	1,004	86.25%	**42.79%**	25.18%	19.94%
Exp. #7	99,94%	1,838	4,397	1,004	86.13%	**42.36%**	25.39%	19.93%
Exp. #8	99.77%	1,816	4,337	997	86.05%	**41.80%**	25.39%	19.69%
Exp. #9	99.55%	1,824	4,344	994	86.12%	**41.90%**	25.46%	19.77%

Table 47: Results of the Mixture of Unigrams + Bigrams on Pre-clustering Experiments with Document Frequency

Exp.	Lenient			Strict		
	$F^L_{0.5}$	F^L_1	F^L_2	$F^S_{0.5}$	F^S_1	F^S_2
Exp. #1	**75.84%**	**63.27%**	**54.27%**	**21.81%**	**21.16%**	**20.54%**
Exp. #2	**74.29%**	**62.28%**	**53.61%**	**21.55%**	**21.01%**	**20.48%**
Exp. #3	**75.58%**	62.55%	53.36%	22.20%	21.37%	20.61%
Exp. #4	**75.44%**	62.55%	53.42%	21.88%	21.11%	20.38%
Exp. #5	**71.77%**	57.33%	47.72%	23.83%	22.19%	20.76%
Exp. #6	**71.68%**	57.20%	47.58%	23.92%	22.25%	20.81%
Exp. #7	**71.38%**	56.79%	47.15%	24.07%	22.33%	20.82%
Exp. #8	**71.02%**	56.27%	46.59%	24.01%	22.18%	20.61%
Exp. #9	**71.11%**	56.38%	46.70%	24.08%	22.26%	20.70%

Table 48: F-Measures of the Mixture of Unigrams + Bigrams on Pre-clustering Experiments with Document Frequency

Exp.	**Sim_T**	**n**	**DF**	**TV**	**Expansion**	**Nb of Features**
Exp. #1	0	20	0	0.98	Yes	2,825
Exp. #2	0	50	0	0.98	Yes	2,638
Exp. #3	0	100	0	0.98	Yes	2,536
Exp. #4	0	500	0	0.98	Yes	2,373
Exp. #5	0	1,000	0	0.98	Yes	2,330
Exp. #6	0	2,000	0	0.98	Yes	2,084
Exp. #7	0	3,000	0	0.98	Yes	1,838
Exp. #8	0	4,000	0	0.98	Yes	1,648
Exp. #9	0	5,000	0	0.98	Yes	1,516

Table 49: Description of the Mixture of Unigrams + Bigrams on Pre-clustering Experiments with Term Variance Frequency

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	**P^L**	**R^L**	**P^S**	**R^S**
Exp. #1	100.00%	1,859	3,507	216	96.13%	**36.45%**	33.30%	20.15%
Exp. #2	100.00%	1,848	3,537	218	96.10%	**36.58%**	32.98%	20.03%
Exp. #3	100.00%	1,843	3,527	224	96.00%	**36.48%**	32.94%	19.98%
Exp. #4	**100.00%**	**1,835**	**5,306**	**225**	**96.94%**	**48.51%**	**24.91%**	**19.89%**
Exp. #5	**100.00%**	**1,834**	**5,330**	**213**	**97.11%**	**48.67%**	**24.86%**	**19.88%**
Exp. #6	**99.99%**	**1,839**	**5,277**	**233**	**96.82%**	**48.34%**	**25.02%**	**19.94%**
Exp. #7	**99,94%**	**1,836**	**5,323**	**213**	**97.11%**	**48.64%**	**24.90%**	**19.91%**
Exp. #8	99.77%	1,805	5,217	1,104	86.41%	**47.71%**	22.21%	19.57%
Exp. #9	99.55%	1,824	5,272	1,096	86.62%	**48.21%**	22.26%	19.77%

Table 50: Results of the Mixture of Unigrams + Bigrams on Pre-clustering Experiments with Term Variance Frequency

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**72.42%**	52.86%	41.62%	29.46%	25.11%	21.88%
Exp. #2	**72.51%**	52.99%	41.76%	29.20%	24.93%	21.74%
Exp. #3	**72.38%**	52.87%	41.64%	29.16%	24.87%	21.69%
Exp. #4	**80.81%**	**64.67%**	**53.90%**	**23.71%**	**22.12%**	**20.73%**
Exp. #5	**80.99%**	**64.84%**	**54.06%**	**23.67%**	**22.09%**	**20.71%**
Exp. #6	**80.65%**	**64.49%**	**53.72%**	**23.81%**	**22.19%**	**20.78%**
Exp. #7	**80.97%**	**64.81%**	**54.03%**	**23.71%**	**22.12%**	**20.74%**
Exp. #8	**74.35%**	61.47%	52.40%	21.62%	20.80%	20.04%
Exp. #9	**74.71%**	61.94%	52.90%	21.71%	20.94%	20.23%

Table 51: F-Measures of the Mixture of Unigrams + Bigrams on Pre-clustering Experiments with Term Variance Frequency

Unigrams on Latent Semantic Analysis Experiments

Similarly the clustering experiments, unigrams only is chosen as feature to represent snort messages and CAPEC attack fields. Tables 52, 53, 54 and 55 show the mapping quality in terms of term frequency and tf.idf.

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	99.46%	106	4,243	110	97.53%	**29.54%**	2.37%	1.15%
Exp. #2	99.46%	100	5,246	110	97.98%	**36.32%**	1.83%	1.08%
Exp. #3	**99.46%**	**1,796**	**5,241**	**110**	**98.46%**	**47.81%**	**32.94%**	**25.12%**
Exp. #4	**99.48%**	**1,797**	**5,242**	**122**	**98.29%**	**47.82%**	**25.09%**	**19.48%**
Exp. #5	**99.49%**	**1,797**	**5,236**	**122**	**98.29%**	**47.78%**	**25.11%**	**19.48%**
Exp. #6	**99.50%**	**1,797**	**5,233**	**122**	**98.29%**	**47.76%**	**25.12%**	**19.48%**
Exp. #7	99.52%	1,797	5,233	122	98.29%	47.76%	25.12%	19.48%

Table 52: Results of the Unigram on Latent Semantic Analysis Experiments with Term Frequency

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**66.79%**	45.35%	34.33%	1.95%	1.54%	1.28%
Exp. #2	**73.14%**	52.99%	41.55%	1.61%	1.36%	1.18%
Exp. #3	**81.24%**	**64.36%**	**53.29%**	**23.75%**	**21.94%**	**20.39%**
Exp. #4	**81.16%**	**64.34%**	**53.29%**	**23.72%**	**21.93%**	**20.39%**
Exp. #5	**81.14%**	**64.30%**	**53.25%**	**23.74%**	**21.94%**	**20.40%**
Exp. #6	**81.12%**	**64.28%**	**53.23%**	**23.75%**	**21.94%**	**20.40%**
Exp. #7	**81.12%**	64.28%	53.23%	23.75%	21.94%	20.40%

Table 53: F-Measures of the Unigram on Latent Semantic Analysis Experiments with Term Frequency

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	99.46%	4	3,448	108	96.96%	**23.45%**	0.11%	0.04%
Exp. #2	99.46%	1,702	895	1,093	70.38%	**17.64%**	46.12%	18.45%
Exp. #3	**99.46%**	**1,764**	**1,796**	**21**	**99.41%**	**24.18%**	**49.25%**	**19.12%**
Exp. #4	**99.48%**	**1,787**	**1,795**	**2**	**99.94%**	**24.33%**	**49.86%**	**19.37%**
Exp. #5	**99.49%**	**1,788**	**1,795**	**1**	**99.97%**	**24.34%**	**49.88%**	**19.38%**
Exp. #6	**99.50%**	**1,790**	**1,795**	**1**	**99.97%**	**24.35%**	**49.91%**	**19.41%**
Exp. #7	99.52%	1,790	1,795	1	99.97%	24.35%	49.91%	19.41%

Table 54: Results of the Unigram on Latent Semantic Analysis Experiments with TF.IDF

Exp.	Lenient			Strict		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**59.65%**	37.82%	27.69%	0.08%	0.06%	0.05%
Exp. #2	**44.04%**	28.21%	20.75%	35.48%	26.36%	20.97%
Exp. #3	**61.29%**	**38.90%**	**28.50%**	**37.45%**	**27.55%**	**21.79%**
Exp. #4	**61.64%**	**39.14%**	**28.67%**	**37.92%**	**27.90%**	**22.07%**
Exp. #5	**61.66%**	**39.15%**	**28.68%**	**37.94%**	**27.92%**	**22.08%**
Exp. #6	**61.67%**	**39.17%**	**28.69%**	**37.97%**	**27.95%**	**22.11%**
Exp. #7	**61.67%**	39.17%	28.69%	37.97%	27.95%	22.11%

Table 55: F-Measures of the Unigram on Latent Semantic Analysis Experiments with TF.IDF

Bigrams on Latent Semantic Analysis Experiments

Besides using unigrams as the feature, Tables 56, 57, 58, 59 and 60 below show the results when using bigrams only with latent semantic analysis.

Exp.	$\mathbf{Sim_T}$	n	DF	TV	Expansion	Nb of Features
Exp. #1	0	100	0	0	Yes	73,350
Exp. #2	0	500	0	0	Yes	73,350
Exp. #3	0	1,000	0	0	Yes	73,350
Exp. #4	0	2,000	0	0	Yes	73,350
Exp. #5	0	3,000	0	0	Yes	73,350
Exp. #6	0	4,000	0	0	Yes	73,350
Exp. #7	0	5,000	0	0	Yes	73,350

Table 56: Description of the Bigram on Latent Semantic Analysis Experiments

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	99.44%	1,809	3,418	0	100.00%	**35.51%**	34.60%	19.61%
Exp. #2	**99.45%**	**1,793**	**3,460**	**0**	**100.00%**	**35.69%**	**34.13%**	**19.44%**
Exp. #3	**99.45%**	**1,793**	**3,460**	**3**	**99.94%** %	**35.69%**	**34.11%**	**19.44%**
Exp. #4	99.47%	1,794	3,418	1	99.98%	**35.41%**	34.41%	19.45%
Exp. #5	99.47%	1,793	3,418	0	100.00%	**35.41%**	34.41%	19.44%
Exp. #6	99.49%	1,793	3,418	0	100.00%	**35.40%**	34.40%	19.44%
Exp. #7	99.50%	1,791	3,417	1	99.98%	**35.38%**	34.38%	19.42%

Table 57: Results of the Bigram on Latent Semantic Analysis Experiments with Term Frequency

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**73.35%**	52.41%	40.77%	30.01%	25.03%	21.47%
Exp. #2	**73.50%**	**52.60%**	**40.95%**	**29.65%**	**24.77%**	**21.27%**
Exp. #3	**73.49%**	**52.60%**	**40.95%**	**29.64%**	**24.77%**	**21.27%**
Exp. #4	**73.26%**	52.30%	40.66%	29.82%	24.85%	21.30%
Exp. #5	**73.26%**	52.29%	40.65%	29.81%	24.84%	21.29%
Exp. #6	**73.26%**	52.29%	40.65%	29.81%	24.84%	21.29%
Exp. #7	**73.24%**	52.27%	40.63%	29.79%	24.82%	21.27%

Table 58: F-Measures of the Bigram on Latent Semantic Analysis Experiments with Term Frequency

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	**99.44%**	**1,708**	**3,417**	**0**	**100.00%**	**34.82%**	**33.32%**	**18.52%**
Exp. #2	99.45%	1,793	1,575	0	100.00%	**22.88%**	53.23%	19.44%
Exp. #3	99.45%	1,576	3,277	1	99.97%	**32.97%**	32.46%	17.08%
Exp. #4	99.47%	1,740	1,577	0	100.00%	**22.53%**	52.45%	18.86%
Exp. #5	99.47%	1,763	1,735	0	100.00%	**23.76%**	50.40%	19.11%
Exp. #6	99.49%	1,780	1,728	0	100.00%	**23.83%**	50.74%	19.30%
Exp. #7	99.50%	1,780	1,746	1	100.00%	**23.95%**	50.48%	19.30%

Table 59: Results of the Bigram on Latent Semantic Analysis Experiments with TF.IDF

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**72.76%**	**51.65%**	**40.04%**	**28.73%**	**23.81%**	**20.32%**
Exp. #2	**59.73%**	37.24%	27.05%	39.50%	28.48%	22.26%
Exp. #3	**71.08%**	49.59%	38.07%	27.51%	22.39%	18.87%
Exp. #4	**59.26%**	36.78%	26.66%	38.68%	27.75%	21.63%
Exp. #5	**60.91%**	38.40%	28.04%	37.97%	27.72%	21.82%
Exp. #6	**61.00%**	38.49%	28.11%	38.27%	27.96%	22.03%
Exp. #7	**61.16%**	38.65%	28.25%	38.15%	27.92%	22.02%

Table 60: F-Measures of the Bigram on Latent Semantic Analysis Experiments with TF.IDF

Mixture of Unigrams + Bigrams on Latent Semantic Analysis Experiments

Lastly, the mixture of unigrams and bigrams is used to represent snort messages and attack fields. With the latent semantic analysis (LSA). Tables 61, 62, 63 and 64 indicate the mapping quality.

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	99.46%	105	4,317	109	97.59%	**30.04%**	2.31%	1.14%
Exp. #2	99.46%	1,811	4,359	999	86.06%	**41.92%**	25.26%	19.63%
Exp. #3	99.48%	1,808	5,254	1,017	87.41% %	**47.98%**	22.37%	19.60%
Exp. #4	99.47%	1,794	3,418	1	99.98%	**35.41%**	34.41%	19.45%
Exp. #5	99.47%	1,810	5,254	109	98.48%	**47.99%**	25.23%	19.62%
Exp. #6	**99.50%**	**1,812**	**5,254**	**122**	**98.30%**	**48.00%**	**25.20%**	**19.64%**
Exp. #7	**99.52%**	**1,811**	**5,254**	**122**	**98.30%**	**48.00%**	**25.19%**	**19.63%**

Table 61: Results of the Mixture of Unigrams + Bigrams on Latent Semantic Analysis Experiments with Term Frequency

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**67.32%**	45.94%	34.87%	1.91%	1.52%	1.26%
Exp. #2	**71.09%**	56.38%	46.71%	23.89%	22.09%	20.55%
Exp. #3	**66.47%**	50.57%	40.82%	26.17%	23.25%	20.91%
Exp. #4	**73.26%**	52.30%	40.66%	29.82%	24.85%	21.30%
Exp. #5	**81.36%**	64.53%	53.47%	23.86%	22.07%	20.53%
Exp. #6	81.27%	**64.51%**	**53.48%**	**23.85%**	**22.08%**	**20.55%**
Exp. #7	81.27%	**64.50%**	**53.47%**	**23.84%**	**22.07%**	**20.54%**

Table 62: F-Measures of the Mixture of Unigrams + Bigrams on Latent Semantic Analysis Experiments with Term Frequency

Exp.	**MR**	**MQ**			**Lenient**		**Strict**	
		Correct	**Acceptable**	**Incorrect**	$\mathbf{P^L}$	$\mathbf{R^L}$	$\mathbf{P^S}$	$\mathbf{R^S}$
Exp. #1	99.46%	6	3,462	111	96.89%	**23.56%**	0.16%	0.07%
Exp. #2	99.46%	65	3,431	958	78.49%	**23.75%**	1.45%	0.70%
Exp. #3	99.46%	1,771	927	97	96.52%	**18.33%**	63.36%	19.20%
Exp. #4	99.48%	1,740	3,271	3	99.94%	**34.04%**	34.70%	18.86%
Exp. #5	**99.49%**	**1,745**	**3,452**	**0**	**100.00%**	**35.31%**	**33.57%**	**18.92%**
Exp. #6	**99.50%**	**1,747**	**3,437**	**0**	**100.00%**	**35.22%**	**33.69%**	**18.94%**
Exp. #7	99.50%	1,752	2,554	0	100.00%	**29.25%**	40.68%	18.99%

Table 63: Results of the Mixture of Unigrams + Bigrams on Latent Semantic Analysis Experiments with TF.IDF

Exp.	**Lenient**			**Strict**		
	$\mathbf{F^L_{0.5}}$	$\mathbf{F^L_1}$	$\mathbf{F^L_2}$	$\mathbf{F^S_{0.5}}$	$\mathbf{F^S_1}$	$\mathbf{F^S_2}$
Exp. #1	**59.72%**	37.90%	27.76%	0.12%	0.09%	0.07%
Exp. #2	**53.72%**	36.46%	27.60%	1.20%	0.95%	0.79%
Exp. #3	**52.08%**	30.81%	21.87%	43.40%	29.47%	22.31%
Exp. #4	**72.05%**	50.79%	39.21%	29.71%	24.44%	20.76%
Exp. #5	**73.18%**	**52.19%**	**40.55%**	**29.07%**	**24.20%**	**20.73%**
Exp. #6	**73.10%**	**52.09%**	**40.46%**	**29.15%**	**24.25%**	**20.76%**
Exp. #7	**67.40%**	45.26%	34.07%	33.12%	25.90%	21.26%

Table 64: F-Measures of the Mixture of Unigrams + Bigrams on Latent Semantic Analysis Experiments with TF.IDF