

Intrusion Alert Analysis Framework Using Semantic Correlation

by

Sherif Saad Mohamed Ahmed

B.Sc., Helwan University, 2003

M.Sc., Arab Academy for Science, Technology and Maritime Transport , 2007

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

© Sherif Saad Moahmed Ahmed, 2014

University of Victoria

All rights reserved. This dissertation may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.

Intrusion Alert Analysis Framework Using Semantic Correlation

by

Sherif Saad Mohamed Ahmed

B.Sc., Helwan University, 2003

M.Sc., Arab Academy for Science, Technology and Maritime Transport , 2007

Supervisory Committee

Dr. Issa Traoré, Supervisor

(Department of Electrical and Computer Engineering)

Dr. Kin Fun LI, Department Member

(Department of Electrical and Computer Engineering)

Dr. Jens Weber, Outside Member

(Department of Computer Science)

Supervisory Committee

Dr. Issa Traoré, Supervisor

(Department of Electrical and Computer Engineering)

Dr. Kin Fun LI, Department Member

(Department of Electrical and Computer Engineering)

Dr. Jens Weber, Outside Member

(Department of Computer Science)

ABSTRACT

In the last several years the number of computer network attacks has increased rapidly, while at the same time the attacks have become more and more complex and sophisticated. Intrusion detection systems (IDSs) have become essential security appliances for detecting and reporting these complex and sophisticated attacks. Security officers and analysts need to analyze intrusion alerts in order to extract the underlying attack scenarios and attack intelligence. These allow taking appropriate responses and designing adequate defensive or prevention strategies. Intrusion analysis is a resource intensive, complex and expensive process for any organization.

The current generation of IDSs generate low level intrusion alerts that describe individual attack events. In addition, existing IDSs tend to generate massive amount

of alerts with high rate of redundancies and false positives. Typical IDS sensors report attacks independently and are not designed to recognize attack plans or discover multistage attack scenarios. Moreover, not all the attacks executed against the target network will be detected by the IDS. False negatives, which correspond to the attacks missed by the IDS, will either make the reconstruction of the attack scenario impossible or lead to an incomplete attack scenario. Because of the above mentioned reasons, intrusion analysis is a challenging task that mainly relies on the analyst experience and requires manual investigation.

In this dissertation, we address the above mentioned challenges by proposing a new framework that allows automatic intrusion analysis and attack intelligence extraction by analyzing the alerts and attacks semantics using both machine learning and knowledge-representation approaches. Particularly, we use ontological engineering, semantic correlation, and clustering methods to design a new automated intrusion analysis framework. The proposed alert analysis approach addresses many of the gaps observed in the existing intrusion analysis techniques, and introduces when needed new metrics to measure the quality of the alerts analysis process. We evaluated experimentally our framework using different benchmark intrusion detection datasets, yielding excellent performance results.

Contents

Supervisory Committee	ii
Abstract	iii
Table of Contents	v
List of Tables	ix
List of Figures	xi
Acknowledgements	xiii
Dedication	xiv
1 Introduction	1
1.1 Context	1
1.2 Limitations of Intrusion Detection Systems	4
1.2.1 Alerts Flooding	4
1.2.2 False Positives	4
1.2.3 Interoperability Challenge	5
1.2.4 Isolation	6
1.3 Intrusion Alert Analysis	7
1.4 Research Problem	8
1.5 General Approach	10

1.6	Research Contributions	11
1.7	Dissertation Organization	13
2	Related Work	14
2.1	Alert Verification	14
2.1.1	Techniques based on Environmental Awareness	15
2.1.2	Techniques based on Heuristics and Statistical Analysis	16
2.1.3	Limitations of Existing Alert Verification Techniques	17
2.2	Alert Aggregation	18
2.2.1	Single Sensor Alerts Aggregation	19
2.2.2	Multi-Sensor Alerts Aggregation	22
2.2.3	Limitations of Existing Alert Aggregation	25
2.3	Attack Scenario Reconstruction	26
2.3.1	Similarity and Data Mining Techniques	26
2.3.2	Machine Learning Techniques	27
2.3.3	Knowledge-based Techniques	29
2.3.4	Limitations of Existing Alert Correlation Techniques	33
2.4	Summary	35
3	Intrusion Alert Analysis	36
3.1	Terminology	37
3.2	IDS Alert Analysis Challenges	38
3.2.1	Alert Analysis Correctness Challenges	39
3.2.2	Alert Analysis Automation Challenges	44
3.3	Proposed Alert Analysis Framework	47
3.4	Alert Analysis Evaluation	50
3.5	Summary	52

4	Knowledge-Based Alert Analysis	53
4.1	Knowledge-Based System	54
4.2	Ontology and Ontology Engineering	55
4.2.1	What is an Ontology	55
4.2.2	Ontology Engineering	57
4.3	Proposed Intrusion Analysis Ontology	57
4.3.1	Specification	58
4.3.2	Conceptualization	61
4.3.3	Formalization	73
4.3.4	Validation	75
4.4	Reasoning with Ontology	76
4.4.1	Deductive Reasoning	76
4.4.2	Inductive Reasoning	77
4.4.3	Abductive Reasoning	78
4.5	Semantic Analysis and Correlation	78
4.5.1	Ontology-based Semantic Similarity	79
4.5.2	Ontology-based Semantic Relevance	82
4.6	Summary	85
5	Novel Alert Analysis Techniques	87
5.1	Target Network Example	87
5.2	IDS Alert Verification	91
5.2.1	Alerts Context	95
5.2.2	Alert Verification Using Nearest Neighbors Algorithm	98
5.2.3	Alert Verification Using Rule Induction	102
5.3	IDS Alert Aggregation	116
5.3.1	A Lightweight Alert Aggregation Method	119

5.3.2	Alerts Aggregation Using Semantic Similarity	121
5.3.3	Information Loss Metric	124
5.4	Attack Scenario Reconstruction	126
5.4.1	Semantic-based Alerts Clustering	127
5.4.2	Attack Causality Analysis	134
5.4.3	Identifying Missing Attacks and False Negatives	139
5.5	Summary	141
6	Experiments	143
6.1	Benchmark IDS Datasets	144
6.2	Evaluation Results	145
6.2.1	Handling Massive IDS Alerts	145
6.2.2	Performance Comparison Using DARPA IDS Dataset	161
6.3	Summary	174
7	Conclusion	176
7.1	Work Summary	176
7.2	Future Work	178
	Bibliography	180

List of Tables

Table 4.1	Taxonomic Relations and their Properties	63
Table 4.2	Example of Entry in the Relations Dictionary	69
Table 4.3	Example of Entry in the Class Dictionary	72
Table 4.4	Predicate Examples	74
Table 5.1	Description of the hosts in the target network	89
Table 5.2	Attack Semantic Features	96
Table 5.3	Target Semantic Features	97
Table 5.4	Example of labeled raw IDS alerts	98
Table 5.5	Example of unlabeled raw IDS alerts	100
Table 5.6	Semantic distances between unlabeled alert 1 in Table 5.5 and each labeled alert in Table 5.4	101
Table 5.7	Example of alert training set for rule induction	104
Table 5.8	Alert training set after applying the OBRI technique	105
Table 5.9	Unlabeled novel alert example	106
Table 5.10	Mapping Between Alert Verification and Immune System	111
Table 5.11	Raw IDS Alerts before Aggregation	117
Table 5.12	Summarizing Raw Alerts Using One Hybrid Alert	117
Table 5.13	Summarizing Raw Alerts Using Two Hybrid Alerts	117
Table 5.14	Alerts Examples	128

Table 5.15	IDS alerts generated by an FTP vulnerability exploitation attempt.	137
Table 6.1	ISCX Intrusions Properties	146
Table 6.2	Numbers of false positives versus true positives in the ISCX dataset	146
Table 6.3	Alert Verification Using KNN and Semantic Similarity	149
Table 6.4	Alerts Verification Using Ontology and Rule Induction	151
Table 6.5	A Lightweight Alerts Aggregation Using Hill Climbing Approach	152
Table 6.6	Alerts Aggregation Based on Alerts Semantic Similarity	153
Table 6.7	DARPA 2000 DOS1.0 Dataset Statistics	162
Table 6.8	Semantic Similarity Threshold Vectors	163
Table 6.9	DARPA Semantic-based Alert Aggregation Results	164
Table 6.10	Comparison of alerts aggregation approaches using the DARPA 2000 dataset in their evaluation.	167
Table 6.11	DARPA dataset preprocessing statistics	168
Table 6.12	Multi-sensor Alerts Aggregation Evaluation Results	169
Table 6.13	Comparison of Attack Scenario Reconstruction Approaches Using the LLDDOS1.0 Dataset	173

List of Figures

Figure 3.1	Two Redundant Alerts Generated by the Same Snort Sensor	41
Figure 3.2	Snort Alert for a Privilege Escalation Attack Against Send-mail	45
Figure 3.3	Example of alert information expressed in natural language.	46
Figure 3.4	Intrusion Alert Analysis Framework	48
Figure 4.1	Example of Competency Questions Tree	59
Figure 4.2	A Partition of the Attack Taxonomy	62
Figure 4.3	The Alert Concept Graph	64
Figure 4.4	Adding Attack and Target Description to the Concept Graph	65
Figure 4.5	Ontology Upper Level Classes Snapshot	66
Figure 4.6	Attack Diagnosis Relation	68
Figure 4.7	Attack Scenario Relation	69
Figure 4.8	Information-Gathering Attack Ontology (Partial)	81
Figure 4.9	Ontological Relations between Alerts, Attack, Attacker and Target	84
Figure 5.1	Target Network Topology	88
Figure 5.2	Attack Concept Tree Example	90
Figure 5.3	IP Addresses Concept Tree Example	92
Figure 5.4	Asset Concept Tree Example	93
Figure 5.5	Time Concept Tree Example	94

Figure 5.6	Example of alert context features	99
Figure 5.7	Alert Verification Rules	107
Figure 5.8	Example of Alerts Correlation Graph	129
Figure 5.9	Maximum Cliques in an Alerts Correlation Graph	131
Figure 5.10	Prerequisites and consequences for a buffer overflow attack against a FTP service.	134
Figure 5.11	Transforming Alerts Correlation Graph (a) to Attack Scenario Graph (b) using the Attack Causality Relation	137
Figure 6.1	ISCX Alert Grouped By Day and Category.	147
Figure 6.2	ISCX Alert Grouped By Day and Category.	148
Figure 6.3	Alert Correlation Graph for the first Attack Scenario from the ISCX dataset	155
Figure 6.4	Attack Scenario Graph for the First Attack Scenario from the ISCX dataset	156
Figure 6.5	Attack Scenario Graph for ISCX Second Attack Scenario . . .	158
Figure 6.6	Attack Scenario Graph for ISCX Third Attack Scenario . . .	159
Figure 6.7	Alert Correlation Graph for ISCX Fourth Attack Scenario . .	160
Figure 6.8	APCs for single sensor IDS alerts aggregation	164
Figure 6.9	Hybrid alert obtained from the DARPA 2000 dataset; the hybrid alert represents a mstream DDoS Attack.	166
Figure 6.10	APCs for single sensor IDS alerts aggregation	170
Figure 6.11	Ping Sweep Alerts Clique	171
Figure 6.12	Privilege Escalation Alerts Clique	172
Figure 6.13	Reconstructed Attack Scenario Graph	172

ACKNOWLEDGEMENTS

In the name of Allah, the Most Gracious and the Most Merciful

Alhamdulillah, all praises belongs to Allah the merciful for his blessing and guidance. Allah has truly blessed me and I am forever grateful for his blessings. He gave me the strength to reach what I desire.

I would like to thank Dr.Issa Traroé for all the support and encouragement he provided to me during my work under his supervision. Dr.Traoré has been my mentor, colleague, and friend. It would not have been possible to finish my research without his invaluable help of constructive comments and suggestions.

I would like to thank my dissertation committee Dr. Kin Li and Dr. Jens Weber for their precious time and valuable suggestions for the work done in this dissertation.

I would like to acknowledge the financial, academic and technical support of the ISOT Lab, University of Victoria. My thanks also go to all my colleagues at ISOT lab whom I share great memories, specially Bassam Sayed, Marcelo Brocardo, and Alex Hoole. My deepest gratitude goes to my beloved parents; Mr. Saad Mohammed and Mrs. Sherifa Abdel-Khalik, and also to my sisters for their love, prayers, and encouragement.

Finally, I would like to thank my dearest wife Amera for always standing by me, and believing in me. Thanks Amera for giving me the courage and motivation to keep going.

Any errors that may remain in this dissertation, of course, the responsibility is entirely my own.

Sherif Saad, Victoria, BC, Canada

DEDICATION

I dedicate this work to my family

Chapter 1

Introduction

Webster (online dictionary) defines the word intrusion as *the act of wrongfully entering upon, seizing, or taking possession of the property of another*. In the realm of computer networks an intrusion is the act of violating a security policy by an unauthorized entity. Intrusion alert analysis or alert correlation is one of the most active research topics in the field of intrusion detection systems (IDS). IDS alert analysis focuses on interpreting intrusion alerts and extracting attack intelligence.

1.1 Context

The recent growth in computer networks and their applications made them very appealing target for intrusions. An intrusion or an attack ¹ represents an external or internal malicious activity that violates an organization security policy [32, 5, 75]. One of the most popular and severe classes of cybercrimes is network intrusion. In general, this class of attacks is based on the flaws and vulnerabilities that exist in network protocols and software components. Network intrusions can result in loss of

¹In this dissertation, the terms intrusion and attack have the same meaning, thus they are used interchangeably.

confidentiality, loss of integrity, or unavailability of the target. The target could be any resource such as a computer, a network device, or other assets that belong to an individual or an organization.

To protect computers and networks from intrusions we need to detect and understand intrusion attempts. Intrusion detection refers to the set of approaches to detect malicious actions against the target. Intrusion analysis refers to the process of establishing a clear understanding of intrusion occurrences or attempts. The most common approach to detect intrusions is by using an intrusion detection system (IDS), which is a security appliance that can automatically monitor computers and networks to detect intrusion attempts. The most common approach for intrusion analysis is IDS alert analysis or alert correlation. IDS alert analysis or alert correlation is a sub-branch of event correlation.

There are many ways to categorize intrusion detection systems (IDSs). One way is based on the scope of detection, where an IDS can be either a network-based intrusion detection system (NIDS) or a host-based intrusion detection system (HIDS). Network-based IDS monitors network traffic to detect intrusions and malicious activities that are carried over the network. Host-based IDS monitors local host activities and resources such as local processes, system calls, and file systems to detect intrusion attempts. Here it is important to note that some intrusions can only be detected by NIDS while others can only be detected by HIDS. It is also important to mention that the detection of some complex or multistage intrusions requires using both NIDS and HIDS.

Another categorization of IDS, based on the detection approach, identifies three categories of IDS, namely, signature-based IDS, anomaly-based IDS, and specification-based IDS. Signature-based IDS uses an intrusion signature database to detect intrusion attempts. An intrusion attempt occurs if the network traffic or the system calls

match some of the signatures in the database. Anomaly-based IDS learns the normal behaviors of a target system, and then monitors this target for abnormal behaviors, which are flagged as intrusions. Specification-based IDS uses a set of rules to decide if a set of actions violates a specification of how the system should work. It considers any violation of the system specification as an intrusion attempt. Each of these categories has its strengths and weaknesses. More details about intrusion detection approaches and taxonomies can be found in [51, 10].

An important aspect of an organization protection strategy is to detect malicious behaviors and analyze the intrusion patterns. Usually, organizations deploy firewalls and NIDSs to protect their networks from network intrusions. Firewalls operate over the the network and transport layers. They allow or deny incoming or outgoing traffic using sets of rules. An IDS will analyze the traffic that was allowed by the firewall to go through the network. When the IDS suspects that the network traffic involves an intrusion or a security threat to the target, it raises an alarm. The IDS reports intrusion attempts by generating intrusion alert messages. As we can see, the IDS is only responsible for reporting intrusion attempts but it does not do anything to prevent these attempts.

An extension to IDS, known as intrusion prevention system (IPS), attempts to prevent intrusions. In fact, most IDSs can work as intrusion prevention systems (IPSs). Despite the differences between IDSs and IPSs, both types of systems generate alert messages to report detected or prevented intrusion attempts. In this dissertation we focus on IDS alert analysis and correlation, without making any distinction on the origin of the alerts, whether generated by an IDS or an IPS.

1.2 Limitations of Intrusion Detection Systems

The current generation of intrusion detection systems suffers from several drawbacks that reduce the effectiveness of the intrusion detection process. Here we will focus on the limitations that are related to IDS alert messages. This is because these are the problems that raise the need of IDS alert analysis. In general, there are four major problems related to IDS alert messages, namely, alerts flooding, false positives, lack of interoperability, and isolated alerts, that we discuss in this section.

1.2.1 Alerts Flooding

Alerts flooding is a well known problem in IDS. Because of the rapid growth of network traffic, bandwidth, and size, an IDS sensor may potentially generate a huge number of alerts. For instance, it has been shown in a real case study that a single IDS sensor in an enterprise could generate over 400 alerts per minute and 400,000 alerts on average per day [47], while the network was not really under attack. If the network was under real attack, the IDS could generate hundred of thousands of alerts per hour. Responding to this massive number of alerts in reasonable time is almost impossible and resource-intensive. Even with a large team of intrusion and security analysts, managing alert flooding remains expensive and challenging. Therefore, it is important to develop new techniques that can manage effectively and efficiently alerts flooding.

1.2.2 False Positives

Another major problem with current IDSs is the massive number of false positives generated on a daily basis as shown by several studies [8, 16, 53, 80]. False positives occur when a normal behavior is considered by the IDS as malicious, and a false alert

is generated as a consequence. There are many reasons that can cause the IDS to generate false alerts. For instance, normal behaviors not seen in the learning phase of an anomaly-based IDS system will likely be treated as malicious. A network application that does not follow the Request for Comments (RFC) might seem malicious. Also, a signature-based IDS might use broad or weak signatures that would be triggered by both normal and malicious actions. In addition, a malicious action that is not harmful to the target can result in false alerts. This is because the IDS lacks environmental information such as the target network configuration information. For instance, the lack of environmental information may cause an attack that is relevant only to Windows platform, to trigger an IDS alert, even if the target is a Unix system or patched against this attack.

False positives can significantly decrease the quality of the attack intelligence extracted from the alerts. For example, false positives can result in reconstructing false attack scenarios that never happened while missing the true attack scenarios. Likewise, investigating false positives is a time consuming and expensive process for the intrusion analyst, because rather than focusing on true attacks, the analyst will spend most of his time investigating malicious events that never happened. Therefore, it is important to create effective techniques for reducing the number of false positives. This will allow improving the quality of the extracted attack intelligence and the effectiveness of the attack response and mitigation process.

1.2.3 Interoperability Challenge

The third major problem with IDS alert messages is the lack of interoperability between different IDS sensors. Today with the large variety of attack methods and software available, it is common to use heterogeneous (different types of) IDS sensors to cover the different stages of a typical attack. Alerts generated by heterogeneous

IDS sensors may use different keywords, vocabularies, and formats, which poses an interoperability challenge when it comes to investigating heterogeneous multi-sensor alerts as a response to intrusion attempts. For example, the same attack may trigger an alert generated by a network-based IDS and another alert generated by a host-based IDS. Now, because the two IDS sensors use different vocabularies and formats, the two alerts may look as if they are describing different attacks. This lack of interoperability between the sensors complicate the task of the intrusion analyst during the investigation of intrusion attempts.

Many efforts have been done to support interoperability in heterogeneous IDS. To our knowledge, the greatest effort made so far to address the interoperability problem is the definition of the Intrusion Detection Message Exchange Format (IDMEF). The IDMEF is a common formatting scheme proposed by the Internet Engineering Task Force (IETF) as a solution to address the interoperability challenge in IDSs. However, the IDMEF provides only a syntax for formatting (in a unified way) IDS alerts produced by different IDS sensors. The IDMEF does not provide or specify the keywords and vocabularies used by the alerts to describe the attacks. In other words IDMEF lacks the semantics constructs, which limits its ability to capture the link between similar alerts formatted using syntactically different message structures. Therefore, the IDMEF does not provide a robust solution for the interoperability problem.

1.2.4 Isolation

The last major problem with IDS alerts is isolation. IDSs generate isolated low-level alerts that describe individual attack events. However, most of these low-level alerts are actually related to a larger intrusion pattern that involves either a single intrusion instance or a multistage intrusion. This means that there is some logical relation

between these individual or isolated alerts, which could be explicit or implicit. Intrusion detection systems are not designed to detect or report such relationships between individual alerts, because trying to detect them can be a bottleneck and significantly decrease the IDS performance. However, discovering the relationships between alerts is very important for an intrusion analyst, as these are essential in understanding the intrusion pattern or scenario, and in order to take adequate response.

1.3 Intrusion Alert Analysis

As we mentioned before, an IDS reports intrusion attempts and relies on other security tools and analysis to investigate and respond to these attempts. An intrusion analyst is an individual who has the knowledge and expertise to interpret, investigate, and understand IDS alerts and other security log files. The intrusion analyst investigates IDS alerts to extract attack intelligence, which allows identifying the compromised resources, spotting the system vulnerabilities, and determining the intruder objectives and the attack severity. Using this information, the analyst can define the necessary security policies, take appropriate responses, and design or recommend adequate defensive and preventive strategies.

In general, IDS alert analysis process includes at least four main tasks, namely, alert normalization, alert verification, alert aggregation, and alert correlation. Sometimes the term alert correlation is used to refer to the IDS alert analysis process. An intrusion analyst is responsible for performing these tasks. We describe each of these tasks in the following.

Alert Normalization: consists of converting the alerts generated by heterogeneous IDS sensors into a common format that take into account both the syntax and the semantic of the alerts. This is a required and essential step when dealing

with heterogeneous IDS sensors, and allows addressing the underlying interoperability challenge.

Alert Verification (also known as alert filtering): consists of examining IDS alerts in order to remove false positives. In practice, this may consist of examining the intrusion target to look for any sign of compromise, and then deciding accordingly whether or not an alert should be classified as true or false positive.

Alert Aggregation (also known as alert fusion): consists of grouping similar alerts generated by one or more IDS sensors and summarizing these alerts by generating high-level views of the intrusion attempts. It is important that the alert aggregation does not result in losing important information such as security relevant information. The main objective of alert aggregation is to manage the alert flooding problem and reduce the cost of alert analysis process.

Alert Correlation: consists of finding relevant IDS alerts by discovering implicit and explicit relations between them. Relevant alerts are alerts that belong to the same intrusion pattern or are part of a single multistage intrusion. Alert correlation is used to handle the problem of isolated alerts in intrusion detection systems. Alert correlation allows the reconstruction of intrusion scenario and discovery of intrusion patterns.

As we can see IDS alert analysis is an after the fact process that aims at investigating intrusion attempts and extracting useful intelligence information. Therefore, we can view intrusion alert analysis as an intrusion forensics task.

1.4 Research Problem

IDS alert analysis is a critical process for organizations and IDS users. Unfortunately, IDS alert analysis is a very expensive, time consuming, and resource intensive process.

The currently available IDS alert analysis systems are limited to query engines capabilities without advanced investigation and analysis techniques. As a result, in many cases, IDS alert analysis is performed manually by a team of intrusion analysts. Although there are tools that can assist the human operator in collecting and interpreting the alerts, the task of verifying the existence of malicious activities, establishing underlying scenarios, and identifying their sources is currently based to a large extent on a manual and adhoc process that falls on the shoulder of the human analyst. Considering the massive amount of data to analyze and the different data sources to cover, we can easily understand why IDS alert analysis is a complex and time consuming process. In this context, the automation of the IDS alert analysis becomes a necessity.

Several approaches have been proposed in the research literature to automate the alerts analysis process. However, these approaches are limited to specific intrusion analysis task, such as alert verification, alert aggregation or correlation, and fail to address several major challenges in IDS alert analysis, such as noisy data, uncertainty, novel attack scenarios, etc.

Automating the alert analysis process raises some key challenges as this requires converting the existing ad-hoc approaches into systematic analysis techniques and converting existing expert knowledge into intelligent analysis and decision-making mechanisms. In this dissertation, a new framework for IDS alert analysis using knowledge-based and machine learning approaches is proposed. The proposed framework provides new techniques for alert normalization, verification, aggregation, and correlation. The framework proposed in this dissertation is expected to help intrusion analysts by improving the intrusion analysis process, enhancing the intrusion response or mitigation, and reducing the cost of the intrusion analysis process. The framework addresses in particular a specific set of key intrusion analysis challenges

summarized as follows:

1. Proposing a knowledge representation technique that enables alert messages interoperability between heterogeneous IDS sensors.
2. Defining a robust and systematic technique to describe and measure the similarity between alert messages, that works both for known and novel intrusions.
3. Developing a technique allowing the discovery of implicit and explicit logical relations between isolated alerts that belong to the same intrusion pattern or multistage intrusion.
4. Integrating in a coherent fashion the different tasks involved in a typical intrusion analysis process.

1.5 General Approach

In our opinion, intrusion alert analysis is a process that totally depends on the investigator knowledge and experience. Therefore, we believe that the most promising method to automate this process is by combining knowledge representation (KR) and machine learning techniques to design a robust IDS alert analysis framework. Our approach focuses on taking basic expertise or knowledge about intrusion alerts analysis shared by intrusion analysts and representing such information in a form that is systematic and machine-readable.

The recent developments in semantic web and ontology engineering have opened the door to applying ontology and semantic analysis in the area of intrusion alert analysis. We use an ontology to represent the domain of intrusion analysis, and develop new techniques for intrusion alert analysis using semantic correlation based on semantic similarity, semantic relevance, and semantic reasoning.

On top of the intrusion analysis knowledge base, we introduce several techniques that use this knowledge for automated intrusion alert analysis. To build these new analysis techniques we apply different machine learning and knowledge-representation methods, including example-based learning, clustering, graph mining, and inductive and deductive reasoning.

1.6 Research Contributions

The following key contributions are made in this dissertation:

1. An ontology-based approach to handle intrusion alerts interoperability challenges. The ontology allows us to propose a common IDS alert message format that takes into account both the semantic and syntax of the alert. The use of ontology enables the use of semantic analysis to correlate IDS alerts based on their semantic characteristics. The proposed model can easily be integrated with other IDS standards such as IDMEF. The use of an ontology provides the ability to build an intrusion analysis knowledge-base that is flexible and extensible, and facilitates the integration of the different intrusion alert analysis tasks highlighted earlier. This contribution has been published in two conference papers [66] and [67]
2. A new technique for false positives reduction and alert verification. We proposed two new methods for alert verification. The first method uses alert context and semantic similarity with a nearest neighbors classifier to eliminate false positives. The second method applies a computational model inspired by human immune system and uses ontology and rule induction for alert verification. The proposed technique requires less environmental-awareness in comparison to previous alert verification techniques. This contribution has been published in one conference

paper [81] and a submitted journal paper [71].

3. A new technique for aggregating intrusion alerts and managing alert flooding that measures the semantic similarity between intrusion alerts using new semantic similarity metrics. The proposed technique has several advantages over previous techniques, such as actual alerts reduction and summarization. In addition, we propose a new metric to measure the amount of information loss resulting from aggregating alerts. This allows better evaluation of the alert aggregation process and avoiding loss of security relevant information. None of the existing alert aggregation techniques have dealt with the impact of information loss on the aggregation process. We designed two methods for alert aggregations. The first method applies a hill climbing method to aggregate raw alert based on the taxonomic structure of the intrusion ontology. The second method uses semantic similarity to aggregate alerts. The second method is effective in detecting information loss and avoid losing important security relevant information during the alert aggregation process. This contribution has been published in two conference papers [65, 64], and one journal paper [68].
4. A new intrusion scenario reconstruction technique. The proposed technique applies semantic-based clustering to build alert correlation graph and a clique-based analysis to extract attack patterns from the correlation graph. In addition we proposed a new method for attack causality analysis using attack impact and semantic correlation. Finally, we proposed a new method to tolerate false negatives and predict missing attack steps. This contribution has been published in one conference paper [69] and one journal paper [70].

1.7 Dissertation Organization

The remainder of this dissertation is structured as follows:

Chapter 2 reviews major work in the field of intrusion alert analysis. A discussion of the strengths and weaknesses of related work on alert verification, aggregation and correlation is presented.

Chapter 3 identifies the requirements and challenges of IDS alert analysis process, and then introduces the proposed alert analysis framework.

Chapter 4 introduces our intrusion analysis knowledge-base. The chapter summarizes the requirements involved in designing and developing an intrusion analysis knowledge-base. It also illustrates the use of ontology and semantic correlation for intrusion analysis.

Chapter 5 introduces novel techniques for alert normalization, verification, aggregation, and correlation. It also explains how these new techniques address the challenges and requirements discussed in Chapter 3.

Chapter 6 presents the experimental evaluation of the proposed framework and techniques, by describing the evaluation method and datasets, and discussing the obtained performance results. In addition, it presents the details of the framework prototype and its implementation.

Chapter 7 makes some concluding remarks and outlines some ideas for future work.

Chapter 2

Related Work

Intrusion alert analysis techniques can be categorized into the following three main categories: alert verification, alert aggregation, and alert correlation. Alert verification focuses on classifying alerts as either true positives or false positives. Alert aggregation focuses on managing alerts volume and reducing the effect of the alert flooding problem. Alert correlation focuses on finding related alerts that belong to the same attack pattern or scenario. Several works have been proposed under each category. In this chapter we discuss some of the notable works proposed in each category.

2.1 Alert Verification

Several alerts verification and false positives reduction techniques have been proposed in the literature. Most of the proposed techniques mainly use environmental knowledge to classify alerts as true positives or false positives. Other alerts verification techniques in the literature use heuristics and statistical analysis.

2.1.1 Techniques based on Environmental Awareness

Alerts verification techniques using environmental knowledge can be subdivided into passive and active methods according to how they obtain environmental knowledge. Active methods collect the environmental knowledge directly after the generation of the alert. This means when the alert verification system receives a new alert, the system will start gathering environmental knowledge and use it to decide whether or not the alert is a false positive. On the other hand passive techniques only gather the environmental knowledge statically, usually at the deployment time and then use the gathered information to verify incoming alerts. On one hand, the information collected with active methods always reflect the current state of the target, while the information collected with passive methods can be outdated and less accurate. For example, passive methods will lack information about new services or updates that happened after the deployment. On the other hand active methods are more expensive than passive methods and usually slower than passive methods.

Eschelbeck and Krieger proposed a false positive reduction technique using an active method [24]. The proposed technique combines an IDS sensor, namely Snort, with a vulnerability assessment scanner (named QualysGuard). The verification of an alert generated by the IDS simply consists of using the vulnerability scanning tool to check whether or not the target system is vulnerable to the attack reported by the IDS. If the system is not vulnerable the alert will be considered as a false positive.

Shimamura and Kono proposed a false positive reduction technique using an active method [74] that is built around a new Network Intrusion Detection System (NIDS) that stores, in addition to the attack signatures, knowledge about the behaviors of compromised systems for different types of attacks. When the NIDS detects an intrusion attempt it will delay the alert and monitor the target system for any sign of compromise. An alert will be generated only if the NIDS detects any behavior that

matches the successful attack consequence.

Kruegel and Robertson proposed a false positive reduction technique using an active method [35] that combines the Snort IDS with a Nessus vulnerability scanner. The proposed system can be used for real-time IDS alerts verification, and an attempt is made to minimize the gap between the time of detection by the IDS and the time of verification by the vulnerability scanning tools. This is very important for online alerts verification.

Xiao and Debaio proposed an alert verification technique that combines active and passive methods [89], with the goal of minimizing the cost of the active component. The passive component of the proposed technique consists of a knowledge-base that describes the target network by storing information about the target Operating System (OS), running services, network topology, user account, etc. The active component of the technique consists of the Nessus vulnerability scanner. They classified the alerts into two categories based on whether the corresponding attack require an active method or a passive method. If an alert is classified as a true positive by the passive method, then it will be double checked with the active method. This allows reducing the impact of outdated knowledge base.

2.1.2 Techniques based on Heuristics and Statistical Analysis

A few alert verification techniques do not depend on environmental awareness knowledge directly, and instead, use machine learning and statistical analysis to learn the characteristics of true and false alerts.

The work by Viinikka and colleagues falls under this category. Specifically, the authors proposed a false positive alerts reduction technique by analyzing the time characteristics of alerts stream [86]. Alerts are categorized into trend alerts, periodic alerts and random alerts based on the time characteristics of the alerts, and it is

shown that most of the time trend and periodic alerts are false positives. The proposed theory, however, is illustrated using specific attack signatures, and therefore can hardly be generalized. As a matter of fact, some malicious software can cause an IDS to report alerts in a periodic manner.

Pietraszek proposed an alerts verification technique using machine learning [58]. In the proposed technique, each alert message is represented by a set of features and each message is labeled as either true or false alert. Then the labeled messages are fed into RIPPER, a rule learner algorithm, to construct a rule based classifier that can distinguish between true and false alerts. The main issue with this technique is that the features representing the alerts are very specific (e.g. source IP and destination IP addresses). This is likely to make the classifier very sensitive to specific target network and as a result will limit the ability to use the rules generated by the classifier with other targets.

Ning and colleagues proposed an alerts correlation technique using alerts causality analysis to extract attack scenario and reduce false positives [52]. They assume that true attacks typically trigger more than one attack signatures and therefore the IDS will likely generate several alerts that are causally related. This means that the alerts that cannot be correlated with other alerts are mostly false positive alerts. This assumption might be true for a multi-step attack, but there are many cases (e.g. a denial of service attack such as the land attack) where an attacker can simply attack the system by sending a single or a few packets that will trigger at most one signature.

2.1.3 Limitations of Existing Alert Verification Techniques

As mentioned above, the majority of the existing works on alert verification relies primarily on the use of environmental knowledge to distinguish between false and true positives. However, the reliability of these techniques is questionable because

of their reliance on vulnerabilities scanning tools. Likewise, existing vulnerabilities scanner tools suffer from both false positives and false negatives in their outputs. In other words, the scanning tools can easily provide misleading information that can flaw the verification process. In addition, using an active verification technique that relies on vulnerability scanning tools can be unsafe. In this case, to check whether the system is vulnerable to a certain attack, the scanning tool needs to execute the attack against the target, and this can crash the target or disturb its operations. Moreover, building and maintaining a database of the target system environmental information (e.g. configuration, running services, policies, installed software, updates and patches) or even building a vulnerability database using a vulnerability assessment tool is expensive and often not possible when dealing with large and complex networks.

While most of the existing alert verification techniques rely on environmental awareness, the remaining techniques either entirely ignore environmental information or use assumptions and heuristics about the characteristics of the false positives. Unfortunately, this also results in unreliable alert verifications because it is not possible to define all the heuristics that are necessary to detect all possible false positives. In addition, it is important to be careful when generalizing these heuristics, because the chance of missing true attacks can increase. This represents a serious security threat for the target and decreases the reliability of the alert analysis process.

2.2 Alert Aggregation

A significant amount of papers on alerts aggregation based on single IDS sensor has been produced in the literature. In contrast, only a few papers on multi-sensor alerts aggregation have been published. Single sensor alerts aggregation techniques assume that the alerts have the same format and attributes because they were reported by

the same IDS sensor or sensors from the same vendor. Multi-sensor alerts aggregation take into account the fact that the alerts may have different formats and attributes because they could be generated by different IDS sensors, possibly from different vendors. In this section, we summarize and discuss related works under each of these two categories of alerts aggregation techniques.

2.2.1 Single Sensor Alerts Aggregation

Zhigong proposed a real-time alert aggregation and correlation System [95] that uses five attributes, namely, source IP, source port, destination IP, destination port and intrusion signature. Three metrics are defined to capture attributes similarity. These metrics, however, are very trivial. For instance, one of the metrics, which captures the similarity between intrusion signatures, simply returns 1 if two signatures are equal and zero otherwise. With the proposed technique, alerts based on different intrusion patterns would probably not be aggregated.

Xu and colleagues proposed a graph-based technique to aggregate alerts based on the intrinsic order between them referred to as *happened before* relation [92]. The technique was evaluated with the DARPA 2000 dataset yielding an alerts reduction rate of 64.2%. The main issue with this technique is the high runtime required to construct an alert graph and the assumption of low false positive rate of the IDS which is not always the case in practice.

Hofmann and Sick proposed an online intrusion alert aggregation system [30] in which alerts attributes are divided into two types: categorical attributes and continuous attributes. Examples of categorical attributes are intrusion class, IP address and port number. Examples of continuous attributes are alert time and packet size. Several metrics were defined to capture the similarity between categorical attributes. It is assumed that categorical attributes have a multinomial distribution while continu-

ous attributes have a normal distribution. A maximum-likelihood estimation (MLE) method is used to design a parametrized probabilistic model that clusters or aggregates alerts. Experimental evaluation of the proposed technique with the DARPA dataset and two private datasets yielded alerts reduction rates above 98%.

Wen et al. proposed a lightweight intrusion alert fusion system [88]. The proposed system, called cache-based alert fusion scheme, was inspired from the working mechanism of the CPU cache by applying the concept of Least Recently Used (LRU). The authors believe that the cache-based mechanism can improve the run-time of the aggregation algorithm. Experimental evaluation of the proposed technique with different IDS datasets (DARPA, Treasure hunt and Defcon) yielded an average alert reduction rate of about 91%.

Two other alerts aggregation techniques have been proposed in [85]. The first technique, known as attack thread reconstruction, aggregates a series of raw IDS alerts into a hybrid alert if there is a perfect match between raw alerts attributes, which as mentioned above is limited. Experimental evaluation of this technique using the DARPA 2000 dataset yielded an alerts reduction rate of 6.61%. The second technique, known as attack focus recognition, can aggregate IDS alerts based on different intrusion patterns such as one-to-many or many-to-one attack scenarios. However, the technique cannot aggregate alerts that are the results of the same intrusion attempt but have different intrusion signatures. Experimental evaluation of this second technique yielded an alerts reduction rate of 49.58% with the DARPA 2000 dataset.

Mohamed and colleagues [48, 49] proposed a target centered alerts aggregation technique based on three alert attributes, namely, the destination IP address, the attack signature (type), and the alert message timestamp. An attempt is made to improve the runtime of the technique by comparing the hash value of the attributes values instead of the actual attributes values. A subset of the DARPA dataset was

used to evaluate the technique yielding an alert reduction rate of 86.49%. There are two main problems with this technique. First the aggregated alerts are simply grouped into clusters rather than being converted into hybrid alerts. Therefore, the number of alerts messages is not really reduced with this technique. The second problem is due to the fact that the technique ignores other important alerts attributes such as the source of the attack and the destination port, which will result in the loss of important information required in other alerts analysis tasks such as false positives reduction.

Mahboubian and colleagues proposed an alert verification and aggregation technique inspired by the human immune system [45]. The authors use a set of predefined attack patterns such as one-to-one, many-to-one, and one-to-many to aggregate alerts. After grouping the alerts based on the attack patterns, an artificial immune system combined with a threshold is used to check whether or not the alerts groups relate to dangerous activities. The aggregated alerts that do not trigger the threshold are considered false positives. Experimental evaluation of the approach using the DARPA 2000 dataset yields alerts reduction rates of 97.02% and 98.5% for the LLDOS2.0 and LLDOS1.0 subsets, respectively. No information regarding the false positives reduction rate was provided. The proposed approach suffers from several drawbacks. Firstly, the aggregation is limited to predefined patterns. Secondly, it is possible that there are overlapping alerts between different attack patterns. Thirdly, the fact that alerts verification depends on the attack pattern confidence or threshold introduces risk of misclassification where true alerts are considered as false positives.

Zhuang and colleagues proposed an alerts aggregation technique using a set of similarity metrics to capture the similarity between alerts attributes [96, 90]. Experimental evaluation of the technique yielded an alerts reduction rate of 98.7% with the DARPA 2000 dataset. The proposed technique, however, cannot be used to aggregate

alerts generated by different IDS sensors.

Jie and colleagues proposed an alerts aggregation model that uses binary matching to aggregate alerts, and groups alerts based on whether or not there is a perfect match between their attributes [44]. Evaluation with the DARPA dataset shows that the proposed technique can reach an alerts reduction rate of 90%.

Julisch and colleagues proposed an alerts aggregation technique that uses hierarchical clustering algorithm [33]. Using their own dataset they showed that the proposed approach can reach up to 90% alert reduction rate. This proposed approach has some similarity to our alert aggregation approach we propose in this dissertation. However, there are several features in our approach that distinguish our from the Julisch’s approach. For instance, our approach use an ontological engineering method to build attribute taxonomy, while Julisch’s approach did not describe a systemic method to build the hierarchical structure. Our approach uses semantic clustering using a new semantic similarity metric proposed in this dissertation. Julisch’s approach use the hierarchical structure and shortest path distance to summarize alerts. Therefore, Julisch’s approach in our opinion can overgeneralize the summarized alerts and loss security relevance information. In our alert aggregation approach we proposed a model to measure information loss rate and evaluate the quality of the aggregation process.

2.2.2 Multi-Sensor Alerts Aggregation

To our knowledge, the first multi-sensor alert aggregation technique was proposed by Valdes and colleagues. [84]. The proposed technique uses a similarity function to aggregate alerts that match closely but not necessarily perfectly. Meta alert and alert templates are defined and used to describe IDS alerts. Given a pair of alerts, the similarity function returns for each alert attribute a value between 0 and 1 that reflects

the similarity between corresponding attributes. To deal with different intrusion patterns a set of rules referred to as *Situation-Specific Similarity Expectation* are defined. It is not clear, however, how the authors measure the distance between different intrusion classes. Likewise the proposed technique seems to lack a general mechanism to measure the similarity between different intrusion classes. Evaluation of the technique using a private dataset collected from the lab of the authors, yielded alerts reduction rates between 50%-67%. However, an important limitation of the evaluation process was that while the proposed technique was intended for multi-sensor alerts aggregation only a single IDS sensor was used to generate the alerts involved in the evaluation dataset.

Xu and colleagues proposed an alerts aggregation and fusion technique that can aggregate alerts generated by multiple IDS sensors [91]. The technique uses a multi-keywords scheme to cluster IDS alerts and route clustered alerts to a sensor fusion center (SFC). Each SFC aggregates received alerts based on their source, destination, and attack class. This technique, however, cannot process alerts generated from different intrusion patterns. Although a dataset obtained from the DShield project was used to illustrate the technique, no quantitative performance measure was provided.

Fan and colleagues proposed a distributed IDS alert aggregation technique [25]. In the technique, raw IDS alerts collected from different IDS sensors are first converted to IDMEF format. Then, the converted alerts are processed by an alerts aggregation algorithm that categorizes them into four intrusion classes named *discovery*, *scan*, *DOS*, and *privilege escalation*. For each class of intrusions a similarity function is used to measure the similarity between alerts attributes. Alerts that belong to the same category will be aggregated or fused into meta-alerts. Experimental evaluation of the technique using the DARPA 99 dataset yields an alert reduction rate of about 43.42%.

Debar and colleagues proposed an alerts aggregation and correlation technique for alerts generated by sensors from different vendors [19]. Alerts received from different sensors are expected to be in a standard format such as the Intrusion Detection Message Exchange Format (IDMEF). Four alerts attributes are used for the aggregation, namely, the source, target, alert class, and alert severity. The received alerts are aggregated based on a set of aggregation rules called aggregation situations. Each aggregation rule generates a different meta-alert for the same set of raw IDS alerts, which leads to different aggregation views for the same set of raw IDS alerts. One of the main limitations of the proposed technique is the requirement of perfect match which means that alerts based on different intrusion patterns may not be aggregated. The proposed technique was illustrated only through a case-study. The lack of experimental evaluation meant that no information was provided about the alert reduction rate.

Taha and colleagues proposed a multi-agent system for alerts aggregation and correlation [78] for decentralized IDS architecture. The proposed system consists of a collection of agents. The agent collects the alerts from the different IDS sensors in the network and converts the raw alerts format to the Intrusion Detection Message Exchange Format (IDMEF). In addition, the agent uses a set of rules to handle the alert reformatting process. These rules are initially defined by the administrator. The agent is responsible for choosing the appropriate filter that will process the alerts. The filters are used to aggregate and correlate the alerts based on specific attack patterns. Five filters or attack patterns are used, namely, Fusion, One-to-One, Network-Host, One-to-Many, and Many-to-One. The proposed technique was evaluated using the DARPA 2000 dataset and other public IDS dataset, yielding an average alerts reduction rate between 0.7% and 59.5%.

2.2.3 Limitations of Existing Alert Aggregation

As discussed above, only a small number of multi-sensor alerts aggregation techniques have been proposed in the literature. These techniques mostly use a common format to represent alert messages from different sensors such as IDMEF. However, this only solves the alert message format problem, but cannot ensure that the keywords used by the different sensors to describe the same alert attributes have the same meanings. This of course will limit the performance of the aggregation technique. Likewise, the few existing multi-sensor aggregation techniques either achieved relatively low alert aggregation rates or simply did not report any quantitative performance results. This raises the need of formal alerts representations that consider both the structures and semantics of the alert messages.

Several of the existing alerts aggregation techniques require perfect match of the alerts attributes in the aggregation process. While these techniques do not suffer from information loss, they have very poor performances and do not really address the alert flooding problem. In fact their capability is limited to the elimination of redundant alerts only. On the other hand aggregation techniques that use attribute similarity yield promising performances with alert reduction rates reaching 99% for some techniques. However, none of these techniques consider the quality of the generated hybrid or meta alerts. All the proposed techniques lack an appropriate method to assess the effect of information loss that occur in the aggregated alerts. While the problem of information loss has been pointed out in the literature [13, 30], no metric or technique has been proposed to handle this aspect.

On the other hand, techniques that aggregate alerts by grouping every set of similar alerts into one cluster avoid information loss. However, these techniques do not really reduce the amount of generated alerts, because the number of alerts before the aggregation remains the same after the aggregation. Therefore, these techniques

only perform alert clustering but not alerts reduction, which should be the primary goal of alert aggregation.

2.3 Attack Scenario Reconstruction

As we mentioned before, it is common in the literature to use the term alert correlation to refer to the attack pattern and attack scenario reconstruction. There are two commonly used metrics to evaluate the majority of the proposed techniques in the literature. These two metrics are the *completeness* (also known as the true detection rate) and the *soundness* of the alerts correlation. The two metrics were proposed by Ning et al [52]. Completeness is computed as the ratio between the number of correctly correlated alerts by the number of related alerts (i.e. that belong to the same attack scenario). Soundness is defined as the ratio between the number of correctly correlated alerts by the number of correlated alerts. The completeness metric captures how well we can correlate related alerts together while the soundness metric assesses how correctly the alerts are correlated. Several techniques have been proposed in the literature for attack scenario reconstruction. The proposed techniques fall into one of three main categories based on the type of data analysis methods involved as explained below.

2.3.1 Similarity and Data Mining Techniques

The first category of attack scenario reconstruction techniques use data clustering and data mining methods either to cluster alerts based on their attributes similarity or to mine alerts sequences in specific time interval. Under this category fall the techniques proposed by Li *et al.*, Ding *et al.*, and Al-Mamory and Zhang, respectively.

Li and and colleagues investigated multi-step attack scenario reconstruction using

association rule mining algorithms [40]. The authors assumed that multi-step attacks often happen in a certain time interval and based on this assumption an attack sequence time window is defined and used for association rule mining. The DARPA 2000 dataset was used to evaluate the proposed technique yielding attack scenario detection rate of 92.2%.

Ding and colleagues proposed an attack scenario reconstruction model by extending the apriori association rule mining algorithm to handle the order of intrusion alerts occurrence [22]. The authors introduced, more specifically, a time sequence apriori algorithm for mining intrusion alerts with respect to their order of appearance. The DARPA 1999 dataset was used to evaluate the proposed algorithm. The evaluation results show that the true scenario detection rate is 76% while the soundness of the technique is 53%.

Al-Mamory and Zhang proposed a lightweight attack scenario reconstruction technique by correlating IDS alerts based on their statistical similarity [3]. In the proposed technique, similar raw IDS alerts are grouped into meta-alert (MA) messages. An attack scenario is generated by correlating MA messages using a relation matrix (RM) that defines the similarities between every two MA messages. Using the DARPA 2000 dataset, it was shown that the completeness and the soundness of the proposed technique were 86.5% and 100%, respectively.

2.3.2 Machine Learning Techniques

The second category of attack scenario reconstruction techniques use machine learning methods to learn attack patterns from existing dataset. We found in the literature that few techniques used machine learning methods to reconstruct the attack scenario. Here we cover some these techniques.

Oliver and Cunningham proposed an attack scenario reconstruction technique us-

ing machine learning method based on n-gram analysis [17]. The proposed technique combines alerts produced by one or more heterogeneous IDS sensors into scenarios, and use positive and negative training to build attack scenario membership functions. The scenario membership of a new alert is determined in time proportional to the number of candidate scenarios. The technique was evaluated using a dataset obtained from the DEFCON 8 hacker conference "capture the flag", yielding attack scenario reconstruction accuracy of 88.81%.

Ourston and colleagues proposed a multi-step attack scenarios reconstruction technique using Hidden Markov Model (HMM) [55]. The proposed technique builds one HMM for each attack category involved in the different phases of a multi-step attack. The IDS alerts collected from one or more sensors are stored in a database, and then preprocessed to remove false positives. The preprocessed alerts are assembled into examples to be used by the HMM. Finally, the results of the HMM classification are presented to a human expert who can modify them in case of errors and then store them back into the database. The model was evaluated with a dataset collected by the authors; the results show that the technique can reconstruct correctly 90% of the attack scenarios.

Zhang and colleagues used a fuzzy clustering algorithm coupled with an attack knowledge base for attack scenario reconstruction [94]. The fuzzy clustering algorithm uses several fuzzy distance functions to measure the similarity between alerts signatures, intrusion sources, intrusion targets, application protocols, and intrusions time. Given a set of alerts, the corresponding attack scenario is reconstructed by correlating the alerts based on the prerequisites and the consequences of the attack defined in the attack knowledge base. The intrusion (or attack) prerequisites are the necessary conditions for the intrusion to occur and the intrusion consequences are the outcomes of successful intrusions. The technique was evaluated using the DARPA

LLDOS 2.0 2000 dataset. The performance of this technique was not evaluated using the completeness and soundness metrics. Therefore, it is difficult to compare it to other alert correlation techniques in the literature.

Anbarestani and colleagues proposed an alert correlation technique for attack scenario reconstruction using Bayesian network [7]. The proposed technique uses Bayesian network (BN) model to capture the causal relationship between alerts. The approach consists of generating for a given a set of alerts, all possible orders or sequences of the alerts in this set. Each sequence of alerts represents a candidate attack scenario. Candidate attack scenarios are validated by computing their probability using the BN model and selecting the candidate with the highest probability as the correct attack scenario. The proposed technique was evaluated using the DARPA 2000 LLDOS1.0 dataset achieving 96.72% completeness and 100% soundness.

2.3.3 Knowledge-based Techniques

The third category of techniques use, in most cases, rules for attack scenario reconstruction, and represent attack scenarios and attack knowledge using formal methods. In addition to rule-based methods, some techniques use expert systems and predefined attack scenario templates to process the IDS alerts and reconstruct the attack scenarios.

Ning and colleagues proposed an attack scenario reconstruction technique by correlating intrusion alerts based on the prerequisites and the consequences of the intrusion [52]. The proposed technique involves the following five components: knowledge base, alert preprocessor, correlation engine, alert correlation graph generator, and graph visualization module. The alert preprocessor processes the raw intrusion alerts and converts them into high-level intrusion alert referred to as hyper alert. The knowledge base contains the intrusion prerequisites and consequences, as well as a

predefined template for hyper alerts. The correlation engine correlates the produced hyper alerts to reconstruct the attack scenario. The alert correlation graph generator converts the attack scenario into a graph structure. Finally, the graph visualization module visualizes the graph representing the attack scenario. The DARPA 2000 DOS 1.0 attack scenario dataset was used to evaluate the proposed technique yielding an equal value for the completeness and soundness of 93.96% .

An attack scenario reconstruction technique based on knowledge representation and expert system was proposed by Ding [21]. The proposed technique uses a rule-based hierarchical model where the rules describe the properties of the attacks. The hierarchical model consists of three main layers: scenario layer, rule layer, and attribute layer. The scenario layer is used to describe the different stages of the attack in abstract form. The rule layer is a formal description of the scenario layer implemented using the CLIPS expert system engine [15]. The rule layer involves two main types of rules named initial rules and clustering rules. The attribute layer contains facts describing the scenarios. The attack scenario is reconstructed by extracting attributes values from the IDS alerts and creating the facts of the attribute layer using the initial rules from the rule layer. Then, the created facts are used in matching clustering rules in the rule layer to reconstruct the attack scenario. The authors did not provide any experimental results about the correctness of the proposed system.

Liu and colleagues proposed a multi-step attack scenario reconstruction technique using predefined attack models [42]. The proposed technique defines attack models that an attacker may follow to break in the system. Each defined attack model follows a general attack pattern involving four phases: probe, scan, intrusion, and goal. The attack scenario reconstruction is executed over three main stages, namely, preprocessing stage, attack graph construction stage, and scenario generation stage. The proposed technique was evaluated using the DARPA 2000 LLDOS1.0 dataset

achieving 87.12% completeness and 86.27% soundness.

Ebrahimi and colleagues proposed an attack scenario reconstruction technique using intrusion prerequisites and consequences [23]. The proposed technique uses five alert attributes, namely, the source IP address, source port, destination IP address, destination port, attack type, and alert timestamp. Based on these attributes, similar alerts are grouped using binary matching into different groups where each group of similar alerts represents a candidate attack scenario. For each candidate attack scenario, a set of rules is used to analyze the causality between the corresponding alerts and reconstruct the attack scenario. The technique was evaluated qualitatively using the DARPA 2000 dataset. Since, no quantitative metrics (e.g. completeness and soundness) was used in the evaluation, it is difficult to compare it to other alert correlation techniques in the literature. The idea of grouping similar alerts into candidate attack scenarios and then processing these candidate attack scenarios to reconstruct the attack scenario enhances the efficiency of the reconstruction process by reducing the cost of the attack causality analysis. However, the use of simple similarity metrics such as binary match reduces the effectiveness of the technique and limits it to simple attack scenarios.

Yan and colleagues proposed FAR-FAR as a frame-based and first-order logic technique for attack intelligence gathering [93]. The FAR-FAR technique represents IDS alerts in normalized and semantic form, and uses backward-chaining reasoning using semantic rules to reconstruct attack scenarios. It is based on four stages: aggregation, normalization, correlation, and visualization. For each collection of alerts generated by the same sensor the FAR-FAR technique aggregates the alerts to remove redundant alerts. The aggregated alerts from different sensors are normalized and converted into uniform frame structure using linguistic case grammar and intrusion domain ontology. Then, in the correlation phase the normalized alerts are processed using attack

scenario production rules (collection of if-then statements) and facts obtained from a knowledge base to infer the hidden attack scenario. Finally, the visualization module creates an attack graph to represent the inferred attack scenario. The DARPA 1999 and 2000 datasets were used to evaluate the FAR-FAR technique. The evaluation results show that the technique can reconstruct correctly 92.9% of the attacks scenarios.

Rekhis and Boudrga proposed another formal method-based technique for the reconstruction of attack scenarios [61]. They designed a logic-based digital investigation model using Temporal Logic of Actions (TLA) in which the attack scenario is defined as a series of recurrent and reusable actions. The attack scenario inference process involves the following three phases: initialization phase, forward-chaining phase, and backward-chaining phase. Using both forward and backward chaining, the technique can identify all possible attack scenarios. The forward-chaining identifies a specific attack scenario based on the collected evidences. On the other hand the backward-chaining identifies alternative attack scenarios that lead to the last action in the attack scenario produced in the forward-chaining phase.

Li and colleagues proposed the use of semantic web techniques to design an ontology-based attack scenario reconstruction technique [39]. The knowledge base in the proposed technique contains the attacks prerequisites, attack consequences, and predefined attack scenarios. The attacks knowledge is used to create an alert correlation ontology frame in which the attacks scenarios are represented. The proposed technique focuses only on the representation of the attack knowledge using ontology and other semantic web technologies but it does not describe how to use the knowledge base to identify attacks scenarios during the investigation. The authors mentioned that they will implement a correlation reasoner as part of their future work.

Al-Mamory and Zhang proposed an attack scenario reconstruction technique inspired by lexical analyzers and formal grammar [4]. An attribute grammar is used to construct the attack scenario and represent the causality between the attack steps. The attribute grammar also carries the information about the attack scenarios and, the prerequisites and consequences of the attacks. The proposed technique was evaluated using the DARPA 2000 LLDOS1.0 dataset achieving 96.41% completeness and 100.00% soundness.

2.3.4 Limitations of Existing Alert Correlation Techniques

Attack scenario reconstruction techniques that use similarity and data mining methods can handle large amount of IDS alerts and in general can reconstruct novel and unknown attack pattern. They suffer, however, from several limitations. One of these limitations is the inability of the techniques to reconstruct complex or sophisticated multi-step attack scenarios. This is because similarity and data mining methods cannot detect causality between individual attacks. Another important issue is their proneness to construct incorrect attack scenarios. For instance, the alert clustering process may lead to overlapping alerts clusters. Alerts from the same scenario may end up in different alerts clusters, while alerts from different scenarios may be placed in the same cluster. It is not possible, however, for one alert instance to belong to two different attack scenarios at the same time. Such situation can occur because either an alert actually belongs to one scenario and is falsely clustered into the other scenario, or there is only one real attack scenario, and the reconstruction technique falsely assumes that there are two scenarios.

Attack scenario reconstruction techniques that use machine learning can reconstruct both simple and complex attack scenarios from the alerts stream. However, these techniques are unable to reconstruct novel attack scenarios unseen during the

training phase and they require re-training for each new deployment; likewise these techniques are limited to known attack scenarios. Of course, it is impossible to find a training dataset that covers all possible attack scenarios. This means novel and unknown attacks scenarios can not be detected unless the system is retrained. The cost of learning new attack scenarios is usually expensive with machine learning methods. Also, the risk of overfitting the model can result in a poor attack scenario reconstruction system. Detecting overfitting with complex problems such as attack scenario reconstruction is expensive. Moreover, some of these techniques are not fully automated and require, as a result, significant human supervision. On the other hand, the reliance on predefined attack scenarios helps minimize the number of false attack scenarios generated.

Attack scenario reconstruction techniques that use knowledge-based methods can detect the causality relations between attacks and reconstruct multi-step attacks. These techniques use hard coded knowledge to represent causality relations and attack models or scenarios. This hard coded knowledge has several problems. For instance, some techniques store templates or descriptions for possible attacks scenarios. In this case a minor difference between the attacks and the templates can prevent the construction of the attack scenario. The same problem exists with causality relations since these techniques rely on explicit knowledge. These techniques are unable to detect hidden and implicit relations between attacks, which makes it difficult for them to recognize attack scenarios with implicit causal relationships. For instance, they do not tolerate false negatives (i.e. missing alerts). Another major problem with knowledge-based techniques is related to the implementation and maintenance of the knowledge base. To represent the knowledge required to reconstruct the attacks, most of the techniques rely on complex methods that are not easy to use. This limits the ability of intrusion analysts and security experts to update and maintain the

knowledge-base.

2.4 Summary

In this chapter we presented the state of the art in IDS alert correlation and analysis. In the literature we found that previous works in the area of alert correlation focus on three main tasks, namely, alert verification, alert aggregation, and attack scenario construction. In alert verification, the majority of the proposed techniques apply environmental awareness knowledge to filter alerts and eliminate false positives. Alert aggregation techniques mainly apply extraction methods based on known attack patterns and focus on alerts generated in single sensor environment. Attack scenario construction is the most active problem in alert correlation. Many approaches were proposed in the literature. These approaches applied data mining, machine learning and knowledge-based techniques to analyze alerts and reconstruct attack scenarios. We explained the limitations of existing alert correlation techniques. In the next chapters we present our novel approach to tackle the limitations of existing IDS alert correlation approaches.

Chapter 3

Intrusion Alert Analysis

Intrusion alert analysis is one of the most active research directions in the field of intrusion detection system. In general, there are four main tasks in IDS alert analysis, namely, alert normalization, alert verification, alert aggregation, and alert correlation. There are some other tasks, such as attack attribution, attack profiling, intent recognition, and impact analysis but these tasks are less common and greatly depend on the other four main tasks. Actually, the majority of the research contributions in alert analysis focus on alert correlation, and more specifically, on attack plan recognition and multistage attack scenario reconstruction.

It is common in the literature to use the term alert correlation to refer to the entire alert analysis process. This may be because alert analysis is a sub-category of event correlation. The field of alert analysis lacks common terminologies, which can hinder collaboration between researchers.

In this chapter we introduce the terminologies we are using in our research to avoid possible confusion. Then, we revisit the main challenges in IDS alerts analysis. We focus on the challenges that affect the correctness and the automation of the alert analysis process. At the end of this chapter we propose an IDS alert analysis

framework that can handle the alert analysis challenges.

3.1 Terminology

The field of IDS alert analysis is filled with inconsistent terminologies. This makes it difficult to discuss, understand, and relate the various works published on this topic. For this reason we introduce in this section the terminologies used through this dissertation.

Attack: an event, that violates an organization security policy through the exploitation of some existing vulnerability. An attack can be either an intrusion (event detected in the inbound traffic) or an extrusion (event detected in the outbound traffic). In practice, the term intrusion is used to refer to both intrusion due to inbound traffic and extrusion due to outbound traffic. Similarly, we will use in the rest of the thesis the term intrusion to refer interchangeably to refer to either type of attacks".

Attack Pattern: a collection of events or actions that are known to be taken generically by an individual with malicious intent to execute an attack.

Alert: a message generated by the IDS to report an intrusion/extrusion attempt. Each alert message is expressed by a set of n alert attributes that convey specific information.

Alert Attribute: a field in the alert message that describes a characteristic of the intrusion reported by the alert, such as the source IP address or port of the intrusion, or the time or type of intrusion. The value of the alert attribute could be numerical or symbolic; the majority of alert attributes values are symbolic data.

Alert Fingerprint: the set of alert attributes that are sufficient to distinguish between different alerts. This set of alert attributes usually contains the intrusion source and destination addresses, and the intrusion type and timestamp. In multi-IDS en-

vironment, additional information such as the IDS sensor name or ID may be added to this set of attributes.

Normalized Alert: any raw IDS alert formatted using a common format that considers both the syntax and the semantic of the alert attributes.

Verified Alert: is a normalized alert that has been validated using one or more alert verification technique. A normalized alert can be a true or false alert. On the other hand a verified alert is an alert that the analyst or the alert analysis system believes to be a true alert.

Raw Alert: an alert generated directly by a specific IDS sensor. In other words it is an alert described using the default format and vocabularies of the IDS sensor, before any normalization or preprocessing.

Alerts Stream: a collection or a sequence of intrusion alerts generated by heterogeneous IDS sensors. Usually an alerts stream consists of raw IDS alerts.

Hybrid Alert: a high-level (in terms of abstraction) alert message that represents and summarizes a set of normalized alerts. Hybrid alerts are obtained by aggregating low level (or more specific) alerts.

3.2 IDS Alert Analysis Challenges

The main objective of alert analysis techniques is to automate as much as possible any of the tasks involved. This means designing for each task a mechanism that can perform one or more alert analysis tasks correctly with a minimum human supervision or assistance. In general, there are two categories of challenges that limit the ability to automate intrusion alert analysis. The first category refers to challenges that affect the correctness of the alert analysis techniques. The second category refers to challenges that affect the automation of the alert analysis techniques.

3.2.1 Alert Analysis Correctness Challenges

The first category of alert analysis challenges consists of a series of challenges that affect the correctness of the output of the alert analysis. These include challenges related to noisy data, decision under uncertainty, and novel attack patterns.

Impact of Noisy Data

Usually false positives and redundant alerts are the main sources of noisy data.

An alert is generated by the IDS because either it has detected an actual attack (a true positive) or it is considering a normal event as an attack (a false positive). A false positive could also be generated because an unsuccessful attack has occurred. For instance, a code red attack can make the IDS generate one or more alerts even if the target Operating System is incompatible with the attack. For instance, an alert triggered by a code red attack on a Unix system is most likely a false positive since such attack is only relevant to Windows platform. Although, some organizations could be interested in knowing about unsuccessful attacks against their networks, alerts triggered by such attacks are in general categorized as false positives.

It is essential to prune the alerts to remove noisy data such as false positives. However, filtering false positives is very challenging for the following reasons:

1. It requires information about the target system configuration, running services, security policy, etc.
2. It requires understanding the attack impacts and outcomes, and its relations to other attacks under investigation.

If poorly designed, a false positive filtering technique could fail to eliminate a significant number of false positives or incorrectly consider true alerts as false positives. In both cases, this will affect negatively the attack intelligence gathering and the

attack scenario reconstruction. The occurrence of false positives means that the intrusion analyst will spend time investigating attacks that did not occur or reconstruct false attack scenarios. At the same time, incorrectly flagging true alerts as false positives and consequently removing them will result in reconstructing incomplete attack scenarios and missing important attack intelligence.

As mentioned above, besides false positives, redundant alerts represent the second type of noise in IDS alert analysis. In general an alert a_j is a duplicate or redundant instance of another alert a_i , if a common subset of the attributes of a_i and a_j have the same values. Usually this subset of attributes includes the source of the attack, the destination of the attack, and the attack signature or type. We will call this subset of alert attributes the *alert key attributes*.

There are three types of redundant alerts. The first type occurs when several alerts are triggered by the same malicious event. For example, a brute force password attack against an online account could generate several alerts triggered by the different login attempts. This type of redundant alerts can easily be eliminated by grouping the redundant alerts into a hybrid alert that represents the redundant alerts during the attack phase.

The other two forms of redundant alerts are more complex. This is because the redundancy relation between the alerts is not explicit. This happens when the alerts do not have exactly the same key attributes. This could happen, for instance, when the alerts convey the same information but use different data formats or structures. For instance, the Snort IDS version 2.9.2¹ [77] generates for a *land attack*² at least two alerts with two different signatures shown by Figure 3.1.

As we can see from Figure 3.1 a single packet can trigger more than one attack

¹Based on the default Snort rule-set and configuration.

²A land attack is a denial-of-service (DOS) attack where the attacker crafts and sends a packet containing the same source and destination IP address to the victim machine to crash the TCP/IP stack.

```

[116:151:1] “(snort decoder) Bad Traffic Same Src/Dst IP”
[Priority: 3] 05/04-08:53:52.824985 192.168.10.2:0 -> 192.168.10.2:0
UDP TTL:254 TOS:0x0 ID:14733 IpLen:20 DgmLen:33 DF
UDP header truncated

```

(a) Snort Alert for a land attack attempt.

```

[1:527:8] “BAD-TRAFFIC same SRC/DST [Classification: Potentially Bad Traffic]”
[Priority: 2] 05/04-08:53:52.824985 192.168.10.2:0 -> 192.168.10.2:0
UDP TTL:254 TOS:0x0 ID:14733 IpLen:20 DgmLen:33 DF
UDP header truncated

```

(b) Another Snort Alert for the same land attack attempt as in Figure 3.1a.

Figure 3.1: Two Redundant Alerts Generated by the Same Snort Sensor. As we can see the alert in 3.1a and the alert in 3.1b have the same source address (192.168.10.2:0), destination address (192.168.10.2:0), and time stamp (05/04-08:53:52.824985). However, they have different attack signatures, namely, (116:151:1) and (1:527:8). It is clear that the two alerts are redundant but this obvious fact is not easily detected by a machine.

signature and trigger for the same IDS sensor many alerts.

Handling redundant alerts becomes even more complex when those alerts are generated by heterogeneous IDS sensors. In this case, the lack of interoperability between the sensors can make it difficult to detect redundant alerts even for human investigators. For instance, there is a known attack against SGI Telnet servers that come with a default account where the username and password are *4Dgifts*. This attack will be detected and reported by the Bro IDS [56] with the message "*Sensitive Username In Password*", while Snort IDS will generate for the same attack an alert with the message "*TELNET 4Dgifts SGI account attempt*". Even if the two IDSs use IDMEF (format) to report this intrusion they will use their own language/vocabularies to describe the intrusion. It is not clear or obvious how these two messages are related even though they are actually referring to the same intrusion instance.

Failing to process adequately noisy data will cost time and affect the performance of the alert analysis system. Considering that the source of noise in intrusion alerts information consists of redundant alerts and false positives, given an IDS log file, we calculate the amount of noisy data N involved using the following equation:

$$N = |(A \cup B) - (A \cap B)| \quad (3.2.1)$$

Where A is the set of redundant intrusion alerts and B is the set of false positives involved in the log file³. One of the main objectives of an IDS alert analysis system is to minimize the amount of noisy data N as much as possible.

Decision under Uncertainty

The reliability of the output of the alert analysis process is greatly impacted in the presence of uncertain information. Decision under uncertainty occurs when the an-

³ $|X|$ denote the cardinality of set X

analyst has to make a decision based however on incomplete or inaccurate input or information.

uncertain information involved in IDS analysis process is commonplace. For instance, when the analyst tries to decide whether a given alert is a false or true positive, it is very common that some information required for this decision (e.g. target configuration and running services) is inaccurate or outdated. In this case, the analyst does not have all the information he needs to decide the state of the alert. Therefore, this lack of confidence will affect the reliability of the alert analysis process.

Another major source of uncertainty in IDS alert analysis is related to false negatives. False negatives refer to cases where the IDS fails to detect one or more malicious events, which complicate the attack scenario reconstruction process. False positives lead to incorrect attack scenarios, while false negatives either make the reconstruction of the attack scenario impossible or lead to an incomplete attack scenario. The analyst is responsible for considering possible attack steps against the target and classify which ones were potentially missed by the IDS as false negatives. Of course, if there are more than one possible missing event or attack step, this means that there are different possible attack scenarios, in which case, only one scenario should be selected at the end as the reconstructed attack scenario.

Uncertainty also arises in deciding the attack impact. While an IDS can detect the occurrence of an attack, it is not equipped to determine objectively the impact of the attack. Such determination is usually made by the human analyst. Knowing the attack impact is important in reconstructing adequately the attack scenario.

The fact that part of the data used during the alert analysis contains some uncertainties (i.e. inaccurate or incomplete) is a major challenge in alert analysis. Overall, it is essential to develop effective methods to handle missing or inaccurate data in the alert analysis process. These methods should allow the analyst to measure the

confidence of the alert analysis process and reduce the underlying uncertainty.

Novel Attack Pattern or Scenario

The correctness of alert analysis techniques can severely be limited when faced with novel attack scenarios. The problem with novel attack scenarios is that the attacker uses unknown strategy to the analyst to compromise or attack the target. This does not mean that the IDS will fail to detect all the steps involved in the attack scenario. The IDS could even detect all the attack steps without revealing these (to the analyst) because of its novelty.

An attack scenario is novel to the analyst because it contains a set or a subset of attack steps that are not known to the analyst to be part of a single attack scenario. An attack scenario can also be considered as novel if to the analyst knowledge the order or sequence of attack steps do not seem to form a valid attack strategy against the target. This is because some of the relationships (if not all) between the attack steps are unknown to the analyst.

Detecting novelty in IDS alert analysis is challenging because a novel attack scenario often involves implicit relations between corresponding alerts that appear as isolated alerts. Finding these implicit relations is the key to reconstructing novel attack scenario. In addition, besides the challenge posed by attack scenario that are truly novel, it is possible that because of the noise and false negatives, a known attack scenario could appear as a novel one. Therefore, handling adequately noise and uncertainty is a prerequisite for detecting and reconstructing novel attack scenarios.

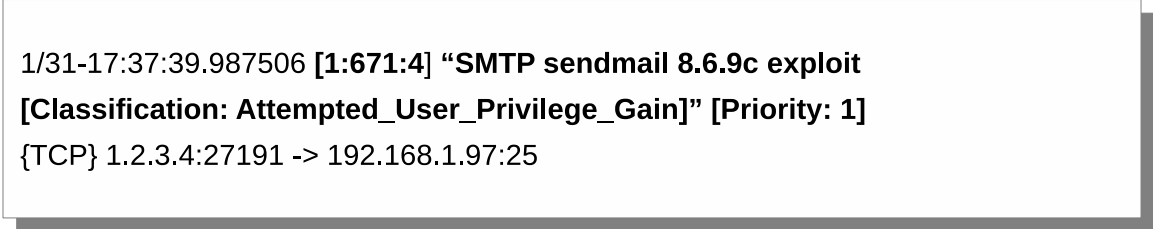
3.2.2 Alert Analysis Automation Challenges

The second category of challenges in IDS alert analysis consist of challenges that limit the ability to automate the alert analysis tasks. These include challenges related to

machine readability and data complexity.

Machine Readability

One of the major difficulties in automating alert analysis is the fact that the alerts are expressed in a format that cannot easily be interpreted by computers. In general, the alert details such as the attack impact, severity, and affected systems are expressed in natural language. Such information is generated primarily toward human consumption and not to be processed by a program. The lack of machine readability of the alerts is worsened by the fact there are no restrictions on the vocabularies used by most intrusion detection systems to express the alerts. It is not unusual that the same IDS sensor uses different vocabularies and keywords to describe the same intrusion attempts. For instance, Figure 3.2 shows an alert generated by Snort to report a privilege escalation attack attempt against a Sendmail server.



```
1/31-17:37:39.987506 [1:671:4] "SMTP sendmail 8.6.9c exploit  
[Classification: Attempted_User_Privilege_Gain]" [Priority: 1]  
{TCP} 1.2.3.4:27191 -> 192.168.1.97:25
```

Figure 3.2: Snort Alert for a Privilege Escalation Attack Against Sendmail

The alert references the attack signature number **1:671** whose description in the Snort (generic) signature database is depicted by Figure 3.3.

As we can see neither the alert information in Figure 3.3 nor the one in Figure 3.2 is machine readable. Moreover, sometimes the attack information may be edited by different intrusion analysts, in which case the same information at some point may become difficult to interpret by other analysts.

Because of the limited machine readability of IDS alerts, it is difficult to design a system that can automatically analyze and extract attack intelligence from the alerts.

Sid: 1:671
Impact: Severe. Remote execution of arbitrary code, leading to remote root compromise.
Affected Systems: Systems running unpatched versions of Sendmail 8.6.10 or earlier.
Corrective Action: Upgrade to the latest version of Sendmail.
Attack Scenario: An attacker sends an email with newline characters and a carriage return, including a path variable of P=/bin/sh. Directives included in the transmission are executed while the message remains in the Sendmail queue.

Figure 3.3: Example of alert information expressed in natural language.

Achieving such objective will require specifying along with alerts their semantics. This semantic specification is important to enable a machine to understand and interpret the meaning of the IDS alerts.

Complex and Symbolic Data

IDS alerts are encoded using complex and symbolic data format. In particular alert attribute values are non-numerical; instead they are based on complex (i.e. composite) and categorical data. Furthermore, while an IDS alert message may consist of a small set of attributes, the knowledge required to analyze the alerts and extract attack intelligence is implicit in this attribute set. For instance, while it is customary to represent explicitly the destination of an attack by an IP address and port number, implicit information (not expressed in the attributes list) such as the platform of the destination host and the service running on the targeted port may be required for adequate analysis of corresponding alert. For instance, the attack signature, which in general is explicitly referenced by the alert, hides more complex data such as the vulnerability exploited in the attack, the system(s) affected by the attack, the impact of the attack and other information related to attack intelligence extraction.

Working with complex and symbolic data is challenging, in particular, when it

comes to measuring the similarity or relevance between alerts, which is very important in alert aggregation and correlation.

The IDS domain is a dynamic domain, the concepts and the data involved in this domain change over time. For instance, new exploits are being created while some existing exploits are becoming obsolete on a regular basis. This means the similarity and the relations between the different concepts in the IDS domain are also dynamic and expected to change over time. Therefore, we cannot assume that the similarity between alerts and concepts in the IDS domain are fixed. This requires designing new methods that can express the similarity between complex data while taking into account conceptual changes in corresponding domain.

3.3 Proposed Alert Analysis Framework

In our opinion an IDS alert analysis framework should support alert normalization, alert verification, alert aggregation and alert correlation. The correctness of any intrusion alert analysis task depends largely on the ability to capture and represent the intrusion analyst knowledge and expertise. Therefore, we believe that the most promising method to automate the different alerts analysis tasks is by combining knowledge representation with machine learning approaches. We use machine learning and knowledge representation to design an intrusion alert analysis framework that can capture and reuse experiences and expertise from intrusion analysts.

The proposed alerts analysis framework consists of the following five components: knowledge base, alert normalization, alert verification, alert aggregation and alert correlation. The framework receives a stream of raw alerts stream as input, analyzes these alerts to eliminate false positives and redundant alerts, summarizes the alerts and reduces the effect of alerts flooding, and finally correlates the alerts to reconstruct

attack patterns and attack scenarios. The overall structure of the proposed alert analysis framework is shown in Figure 3.4.

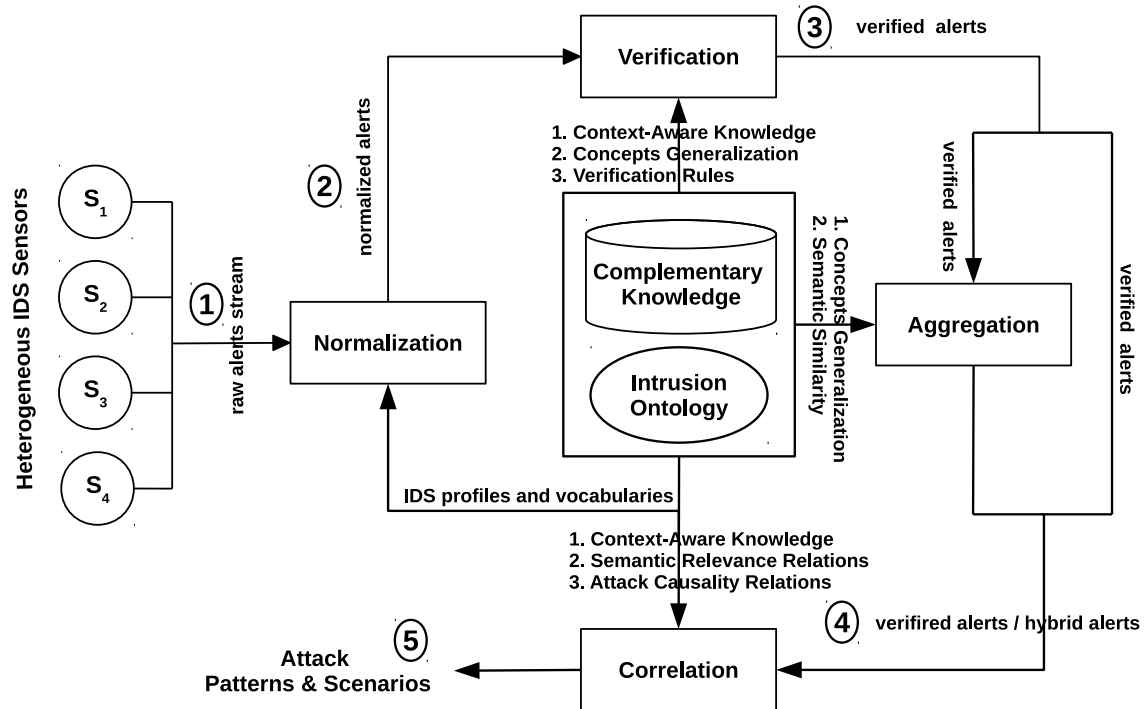


Figure 3.4: Intrusion Alert Analysis Framework

A key component in the proposed alert analysis framework is the intrusion analysis knowledge base, which is used by all the other components of the framework. The intrusion analysis knowledge base is a machine readable knowledge base that consists of an intrusion ontology and complementary knowledge. The intrusion ontology provides semantic specification for the intrusion analysis domain by describing the structure and the semantics of the concepts in the intrusion analysis domain and the different relations between these concepts.

The complementary knowledge is a collection of different intrusion analysis related knowledge such as the IDS profiles, context and environmental knowledge, evaluation metrics, alerts verification and classification rules, and semantic inference rules. The context-awareness is similar to environmental-awareness. However, while

environmental-awareness focuses on describing the network software and hardware assets, context-awareness focuses on network operational knowledge in addition to the network assets. For example, the fact that an employee e_1 is not permitted to access the customer payment records is an example of contextual knowledge

The first phase in our alert analysis framework is the alert normalization. As illustrated in Figure 3.4 the normalization component receives the raw alerts stream as an input and convert the raw alerts to normalized alerts. The raw alerts stream is usually collected from heterogeneous IDS sensors (e.g; Snort, Bro, etc). Each IDS sensor is represented by a profile consisting of a set of formatting rules that define how to convert a raw alert message generated by a given IDS sensor to a normalized alert message that uses a common syntax and semantic based on the intrusion ontology vocabularies.

The alert normalization component uses IDS sensor profiles and the intrusion ontology vocabularies to format the raw alerts into a common format that takes into account the syntax and semantics of the alerts. This phase is important to achieve interoperability and convert the alerts into a machine-readable format. The construction of the IDS sensor profiles and the knowledge-base is not an automated process in our framework. However, the remaining processes such as alert verification, aggregation, and correlation are fully automated.

The second phase in our alert analysis framework is the alert verification, which consists of identifying and eliminating false positives. We use alert context and environmental awareness to identify and eliminate false positives. The proposed techniques use example-based learning and rule-induction. Specifically, from training examples the characteristics of false positives are learned using rule induction. After eliminating the false positives, the output of the alert verification component will consist only of alerts flagged as true positives.

The third phase in our alert analysis framework is the alert aggregation, during which verified alerts are aggregated into hybrid alerts by eliminating redundant alerts and summarizing semantically similar alerts. The semantic similarity between alerts is computed using the intrusion ontology. Alerts that share semantic similarity measure less than a predefined threshold value are fused into the same hybrid alert. An hybrid alert is generated for a set of alerts by replacing corresponding attributes values by more general concepts from the intrusion ontology.

The final phase in our alert analysis framework is the alert correlation. The alert correlation component can take as input either the hybrid alerts generated by the aggregation component or the verified alerts without aggregation. This is because it is possible for one or more hybrid alerts to summarize alerts that belong to different attack scenarios or patterns. Therefore, detecting the causality between the hybrid alert is not possible. In this case the alert correlation component cannot effectively correlate these hybrid alerts. The alert correlation component correlates the alerts by clustering them based on their semantic relevance. The clusters obtained provide the basis to derive the attack patterns and scenarios.

3.4 Alert Analysis Evaluation

Each of the main tasks involved in the proposed alert analysis framework requires different evaluation methods and metrics.

As mentioned before alert normalization is important to enable interoperability and to convert the alert into machine-readable format. To evaluate the performance of an alert normalization technique we need to determine the number of unique alerts generated by each IDS sensor that the normalization technique fails to convert into machine readable format. Such failure is an indication that the knowledge base is

incomplete in terms of vocabularies and concepts. Furthermore, it is possible that an alert is converted to machine readable format while the obtained normalized alert is considered not to be semantically equivalent to the raw alert by an intrusion analyst. In this case the knowledge-base is assumed by the analyst to be inaccurate. Of course, this case is difficult to measure because it relies on the subjective judgment of the analyst.

Alert verification is commonly evaluated by measuring the number of false positives identified by the alert verification technique. However, such metric is limited by the possibility that some true alerts could mistakenly be labeled as false positives during the verification. This means the alert verification can cause false negatives. Therefore, when evaluating an alert verification technique it is important to consider the number of false positive alerts that have been correctly detected as well as the number of true positive alerts that have been mistakenly considered false positives.

In the literature alert aggregation is evaluated based on only the alert reduction rate (ARR). The alert reduction rate is computed as the difference between the original number of alerts and the alerts remaining at the end of the aggregation process over the original number of alerts. Despite its popularity, we believe, however, that the ARR is not enough to evaluate the effectiveness of the aggregation process. In fact, the ARR captures well the quantitative aspect of the alert aggregation process but misses altogether the qualitative perspective. It is important to be able to evaluate if the summarized or the fused version of the alert is not missing important security relevant information from corresponding raw alerts. Likewise, the amount of information loss that result from summarizing the alerts is an important factor in evaluating the alert aggregation as well.

The alerts correlation groups related alerts that belong to the same attack scenario in one cluster. The main factor to evaluate an alert correlation technique is how

accurately the technique can identify related alerts. This means we should consider both the number of alerts that were correctly correlated and the number of alerts that were incorrectly correlated. For incorrectly correlated alerts it is possible that an alert that belong to one attack scenario to be incorrectly correlated into another attack scenario. It is also possible that a truly isolated alert (i.e. not part of an attack scenario or pattern) to be mistakenly correlated with other alerts. It is important to distinguish between these two cases, because, the first case will result in generating two attack scenarios, including one false attack scenario and one true but incomplete attack scenario. While the second case will result in generating only one attack scenario, which is false. Hence, it is important to propose an evaluation method for alert correlation that can measure the confidence of the generated attack scenario and not only the accuracy of the correlation.

3.5 Summary

In this chapter we explained the key challenges in building an IDS alert analysis system. Then, we introduced our alert analysis framework that can tackle these challenges. The proposed framework supports four alert analysis tasks, namely, normalization, verification, aggregation, and correlation. The framework relies on semantic knowledge and ontologies to perform these tasks. We also discussed the available methods for evaluating IDS alert analysis and correlation system and mentioned the different metrics that can measure the performance of alert analysis systems. In the next chapter, we will discuss in detail how we use semantic knowledge and ontology engineering to build a knowledge-based alert analysis system.

Chapter 4

Knowledge-Based Alert Analysis

The intrusion analyst uses prior knowledge to analyze alerts and intrusion attempts. Most of the time, the analyst uses existing knowledge to make inferences about attacks under investigation. For instance, an analyst could infer the impact of a novel attack from the impact of known attacks by finding the most similar known attack to the novel one. Here, the analyst believes that the impact of two similar attacks is mostly the same. What the analyst is doing is to use what he knows, i.e. the "impact of known attacks", to infer what he does not know, i.e. "the impact of a novel attack" given that the novel attack is closely similar to the known attack. The facts and beliefs are the key elements of a knowledge-based system.

In our opinion, the intrusion knowledge representation is a key component to execute any alert analysis task or extract any form of attack intelligence. Likewise, it is an established fact in the literature that the performance of knowledge-based alert analysis techniques is generally better than other techniques that do not rely on knowledge representation. However, knowledge-based alert analysis techniques face two major challenges, namely, the selection of the knowledge representation method, and the adaptability of the knowledge. The first challenge arises as a result of using

either complex or simple knowledge representation method. Maintaining and updating the knowledge becomes very expensive with complex knowledge representation methods. On the other hand, simple knowledge representation methods cannot effectively describe the intrusion analysis domain. In addition, usually the knowledge base is not constructed to support different intrusion analysis tasks. This lack of knowledge adaptability limits the use of the same knowledge base for different alert analysis tasks. As a result, using knowledge-based technique for alert analysis become expensive.

In this chapter we propose a new knowledge base for intrusion alert analysis using ontology. We describe the construction of the proposed ontology, and use semantic analysis and correlation to investigate IDS alert similarity and relevance.

4.1 Knowledge-Based System

The main components of a knowledge-based system are a knowledge base, an inference engine, and a user interface. The knowledge base is a collection of machine-readable concepts related to specific domain. The knowledge is usually collected from one or more domain experts and fed into the knowledge base by a knowledge engineer. The inference engine is the component that mimics the reasoning or the human intelligence allowing the domain expert to solve problems. The inference engine uses the knowledge base to infer logical consequences and derive new conclusions. In other words the inference engine answers queries and questions about the domain described in the knowledge base. The user interface is a human-machine interaction component allowing the user to formulate questions and queries for the knowledge-based system.

The knowledge stored in the knowledge base is a set of vocabularies and facts ranging from simple facts to complex facts about the domain. The first step to build

a knowledge base is to define the vocabularies that will be used to describe the domain. The vocabularies are then used to state facts that describe the domain. The next step is to select a knowledge representation technique to express the knowledge stored in the knowledge base. There are many different knowledge representation techniques such as rules, frames, and semantic networks. There are also many different knowledge representation languages to implement these techniques. In our work, as mentioned above, we use ontologies to build our intrusion alert analysis knowledge base, and as consequence, we use ontologies languages as our knowledge-representation technique and language.

4.2 Ontology and Ontology Engineering

It is only in 2001 that Raskin and Nirenburg have proposed for the first time the use of ontologies in computer and information security [59]. They focused their work on highlighting the advantages of using ontology to represent the domain of information security. Since then, more and more researchers have started to use ontologies in various fields of computer security. Currently, ontologies are being used to develop various security tools such as intrusion detection systems, anti-virus, and other malware detection systems [60, 31].

4.2.1 What is an Ontology

In the field of computer science there are many definitions for the word ontology. Perhaps, the most quoted definition for the term ontology is the one proposed by Gruber [26] as "*an explicit specification of a conceptualization*". In other words, an ontology is a formal representation of a set of concepts, the relations between these concepts in a domain of interest. The basic building blocks of an ontology are classes,

properties (also known as relations), and individuals. The classes are used to represent concepts in a given domain.

Ontologies play a major role in building knowledge representation systems and Artificial Intelligence (AI) systems in general. In fact, ontologies are the core of any knowledge representation system because at minimum they provide the conceptualization of the vocabularies within a specific domain. Of course, without a strong conceptualization we end up with a weak knowledge base that cannot distinguish between concepts within the domain. In this case, reasoning about the domain will be difficult and perhaps useless. Ontologies allow clarifying the structure of knowledge and concepts in the domain which improves reasoning systems.

Ontologies are commonly categorized into lightweight and heavyweight ontologies. This categorization is based on the formality and the granularity of the knowledge represented by the ontology. Heavyweight ontologies allow more complex and meaningful description of a domain of interest by adding axioms and constraints to the ontology. Intelligent systems such as expert and reasoning systems require heavyweight ontologies.

Kruegel and Christopher argued that an ontology for intrusions is a prerequisite for true interoperability between different IDSs [34]. In the last few years, several network intrusion ontologies and taxonomies have been proposed [36, 29, 82, 57, 28, 1]. All of these ontologies can be used to provide common vocabularies and make knowledge shareable by encoding domain knowledge. Hence, they could be used (to some extent) as knowledge bases for an intrusion detection system, but we think they lack the necessary conceptualization for intrusion analysis and attack intelligence extraction.

4.2.2 Ontology Engineering

There are many methods to build an ontology that represents a domain of knowledge and supports reasoning over such knowledge. To build our ontology we used a hyper approach that combines different key features from several ontology development approaches, such as the METHONTOLOGY approach which is based on the work of Lopez and Perez [43]. Most of the ontologies developed today are based on this approach. METHONTOLOGY divides the ontology development process into eleven main tasks. In addition, the process itself is based on evolving prototypes. While we use METHONTOLOGY as our main ontology development approach we also select some key features from other approaches mainly the work of Uschold and King [83] and the work of Gruninger and Fox [27].

Ontology engineering refers to the set of methods and techniques to build and represent the ontologies. Selecting the appropriate methods and techniques to build and represent an ontology depends on the type of the ontology and the intended use of it. There are many knowledge representation paradigms that can be used to represent ontology such as first-order logic, frames, etc. Usually, lightweight ontologies can be represented using software modeling techniques or relational database. Heavyweights ontologies are usually represented using frames or description logic languages.

4.3 Proposed Intrusion Analysis Ontology

Our ontology development lifecycle consists of four main stages, namely, specification, conceptualization, formalization, and verification.

4.3.1 Specification

The purpose of the specification stage is to identify the scope of the ontology and the intended use of the ontology. There are two main ontology engineering activities in the specification stage as follows:

- Specification of the characteristics of the ontology such as its type and knowledge representation paradigm;
- Identification of the scope and the intended use of the ontology.

Our ontology is a heavyweight ontology in terms of formality and granularity and its domain of interest is intrusion analysis. Our intrusion analysis ontology is a method or a task ontology that represents knowledge about the network intrusion domain including concepts and their relations, and attributes and facts about these concepts. Method and task ontologies are special types of ontologies that store problem solving knowledge required to solve a problem or to accomplish a specific task. This means our ontology contains knowledge that represent the network intrusion domain and the knowledge required for intrusion analysis and attack intelligence extraction. The knowledge representation paradigm that we use to formalize our ontology is Description Logic (DL) [11].

To build a method or a task ontology for intrusion analysis, we need to consider the following two types of knowledge:

- *Domain and Factual Knowledge (DFK)*: This type of knowledge includes the concepts of the domain and the conceptual relations within the domain. In other word this type of knowledge contains the sufficient and necessary concepts and relations needed to represent the domain of interest.
- *Problem Solving Knowledge (PSK)*: This type of knowledge includes the knowledge required to perform a specific task or achieve a specific goal using the

domain and factual knowledge. The problem solving knowledge usually consists of a set of complex constraints on the domain and factual knowledge.

We describe the scope and the intended use of our intrusion analysis ontology by creating a set of competency questions that are specific to the intrusion domain and the intrusion analysis. The competency questions are grouped by tasks and structured in hierarchical tree structure or taxonomic structure such that the answer of any parent competency question (usually a complex question) requires the answers from all its children competency questions (less complex questions), as shown in Figure 4.1. Finding the answer of the root question for each competency questions tree means we can simply perform the task correctly. This hierarchical structure helps understanding the dependency between questions and explaining how to find the answer of complex questions by finding the answer of simple ones.

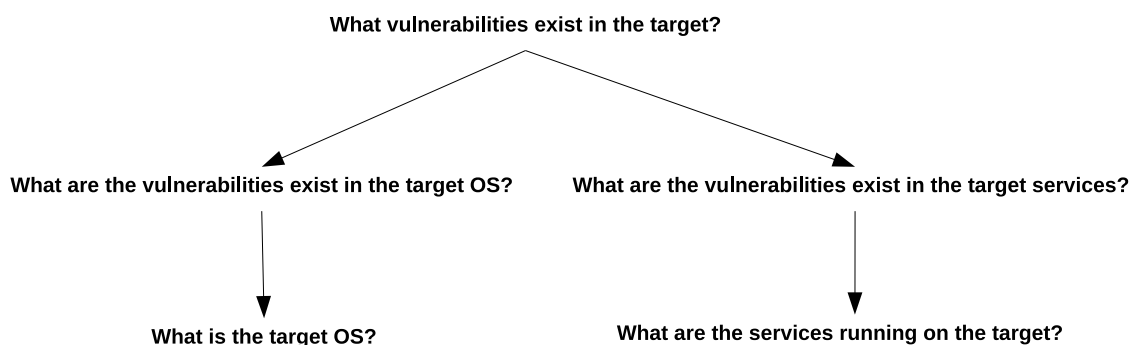


Figure 4.1: Example of Competency Questions Tree

The competency questions can be used to evaluate the ontology by ensuring that the implemented ontology is able to answer all the competency questions identified during the ontology specification stage. Examples of competency questions used to develop our ontology are as follows:

- What vulnerabilities exist in the target system?
- What are the critical attack assets?

- What privileges does the attack require?
- What are the evidences of the attack?
- What are the impacts of the attack?
- Given a set of assets, which assets are vulnerable assets?
- Given a set of privileges, what is the attacker capable of?
- Given a set of attack impacts, what are the attacks that result in these impacts?
- Given an alert, has the compromised host been used to attack other hosts (stepping stones)?
- Given a set of attacks, what evidences are sufficient to prove occurrences of these attacks?
- Given an alert stream, what are the attacks reported by this alert stream?
- Given an alert stream, what are the (attack) targets specified in this alert stream?
- Given an alert stream, what are the attackers specified in this alert stream?
- Given an alert stream, what are the vulnerabilities characterized in this alert stream?
- Given a set of attacks, what are the commonalities between these attacks?
- Given a set of alerts stream, what are the commonalities between these alerts?
- Given a set of vulnerabilities, what are the commonalities between the related attacks?

- Given a set of attacks, what are the vulnerabilities used to execute these attacks?
- Given a set of attacks, what are the impacts of these attacks?

It is important to understand that the ontology stores the knowledge (e.g. classes, relations, constraints) that allow finding the answer to the competency questions, but we need to design the algorithms that can retrieve the answers from the ontology.

4.3.2 Conceptualization

After setting our ontology specification, we move to the conceptualization stage. In this stage we identify the basic concepts or classes, and the relations between these classes in the network intrusion domain. The main ontology engineering activities in the conceptualization stage are to identify the ontology classes, build the classes taxonomy, identify the relations between the classes, describe the classes, and finally define the inference rules.

Ontology Classes

The first activity is to identify the ontology classes. Uschold and King discussed three techniques to identify the classes in the ontology, namely Top-Down, Middle-Out, and Bottom-Up approaches [83]. They recommended the middle-out technique because experience with ontology design shows that middle-out is the most effective one. The middle-out method is very simple to apply. It begins by the key middle-level (not very generalized and not very specialized) classes that exist in the domain, followed by the definition of the more general and more specific classes. However, the middle-out technique does not explain how to define the middle-level classes or the classes that should be included in the ontology. To solve this problem we use the competency questions and their hierarchical structures to decide what are the classes that are

related to the intended use and scope of our ontology.

Using the competency questions we define the key classes in our ontology and then we apply the middle-out technique. For example, we can say that the class *alert* is a middle class in the *evidence* class taxonomy, while the class *evidence* is more abstract and the class *intrusion alert* is more specific. In addition, to cover a wide range of classes in the network intrusion domain, we studied several network intrusion taxonomies and ontologies [36, 29, 82, 57].

The next activity is to build the class taxonomies by grouping similar classes that share the same properties and definitions in the same taxonomy. A taxonomy is a hierarchical structure for the classification or organization of a set of classes that are connected by subclass and superclass relations. Figure 4.2 shows a partition of the attack taxonomy in our intrusion analysis ontology. The root of the attack taxonomy is the class *Attack*.

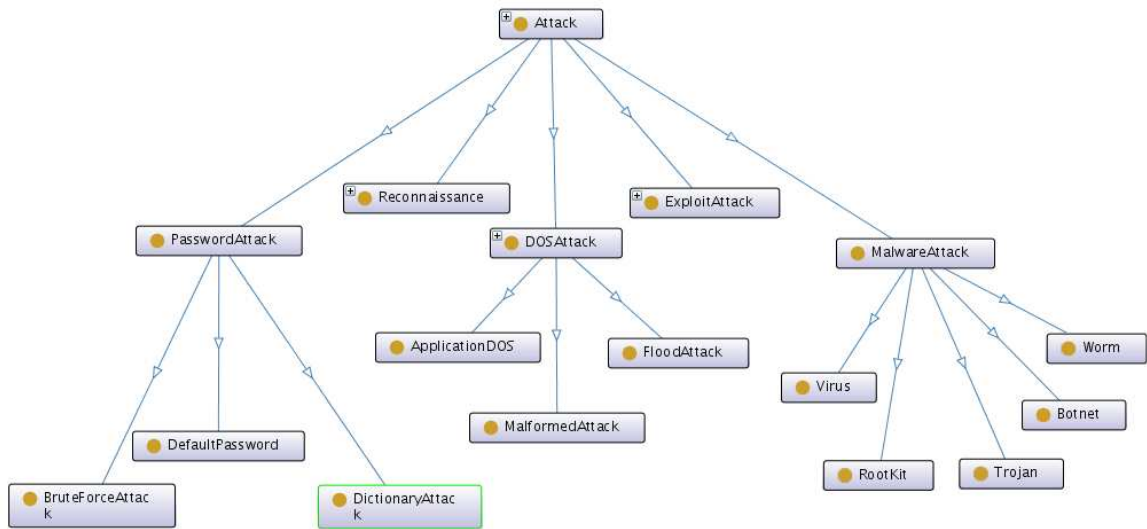


Figure 4.2: A Partition of the Attack Taxonomy

Ontology Relations

The relations between classes in the ontology represent important pieces of information that enable reasoning over the domain. Therefore, the next activity focuses on defining the relations between classes and their characteristics.

There are two types of relations in our intrusion analysis ontology, namely, taxonomic relations and ontological relations. The taxonomic relations are binary relations (2-ary relations) used to categorize classes in our ontology in a taxonomic structure. The taxonomic relations are important for discovering the class subsumption in the ontology. The taxonomic relations in our ontology are listed in Table 4.1.

Relation-Name	Transitive	Reflexive	anti-symmetric
is-A	✓	✓	✓
superclass-Of	✓	✓	✓
subclass-Of	✓	✓	✓
instance-Of	✓	✓	✓

Table 4.1: Taxonomic Relations and their Properties

From Table 4.1 we can see that we use in our intrusion ontology four taxonomic relations. All of these relations are transitive, reflexive, and anti-symmetric. The first relation **is-A** is used to identify the type of a property. For instance, the class *attack* has the property "tool", where "tool is-A Malicious". The **superclass-Of** and **subclass-Of** relations are used to express inheritance. For instance, the class "location" is the superclass-Of "remote-location" and "local-location" and so both "remote-location" and "local-location" are subclass-Of "location". Finally, the **instance-Of** relation is used to link an individual to specific class. For instance, the "code-red" is instance-Of "computer-worm".

Ontological-relations are the relations that link classes from different taxonomies together. These ontological-relations are divided into general relations and specific

relations. General ontological relations are strictly binary relations. On the other hand specific ontological relations are N-ary relations (an N-ary relation is a mapping between a subject and two or more subjects/values).

To define the general ontological relations we use a task-centered technique. We select one or more key classes and define the relations between these classes by constructing a concept graph. Figure 4.3 shows the concept graph obtained from describing the alert concept.

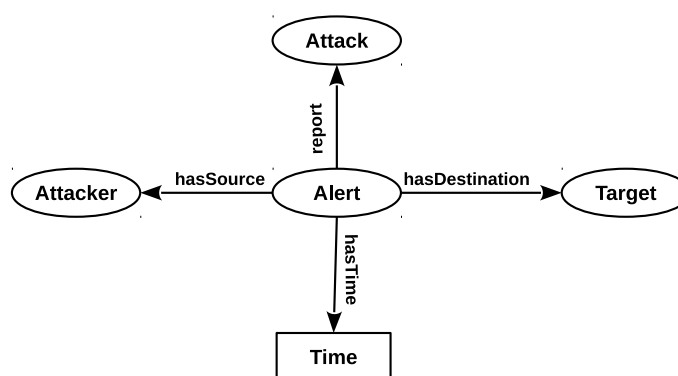


Figure 4.3: The Alert Concept Graph

The relations between the Alert class and the other classes in the concept graph were identified by answering the following competency questions:

- What are the alerts attributes?
- What is the alert source?
- What is the alert destination?
- What is the alert time?
- Which attack was reported by the alert?

In the concept graph in Figure 4.3, a class is shown as an ellipse and a primitive data type as a rectangle. A relation between classes or primitive data types is represented by an arc. The only relations shown in the concept graph are the ones between

the alert and the other concepts. In addition, we need to determine and add to the concept graph any other relations that possibly may exist between the other concepts pairs. Then, we refine the concept graph iteratively by selecting and describing one of the other concepts using the competency questions. For example, if we select the attack (concept) and then target (concept) as the second and the third concepts to describe, respectively, our concept graph will grow as shown in Figure 4.4.

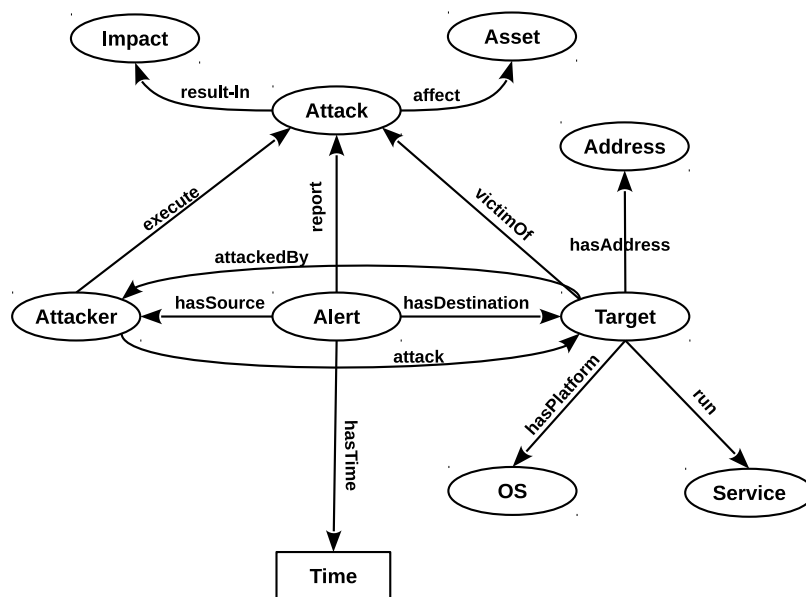


Figure 4.4: Adding Attack and Target Description to the Concept Graph

Every time a new class is added to the graph, the relations between this new class and the existing classes in the concept graph must be identified. Classes are selected (for description) by their order of importance, starting by the most important ones. A class **A** is more important than a class **B** if **A** is shared by more intrusion analysis tasks than **B**. The process ends when all the relations required to answer the competency questions are defined and the intrusion analysis goals are achieved. The upper level classes in our intrusion ontology and the relations between them are illustrated in Figure 4.5.

It is also important to describe the characteristics and properties of the onto-

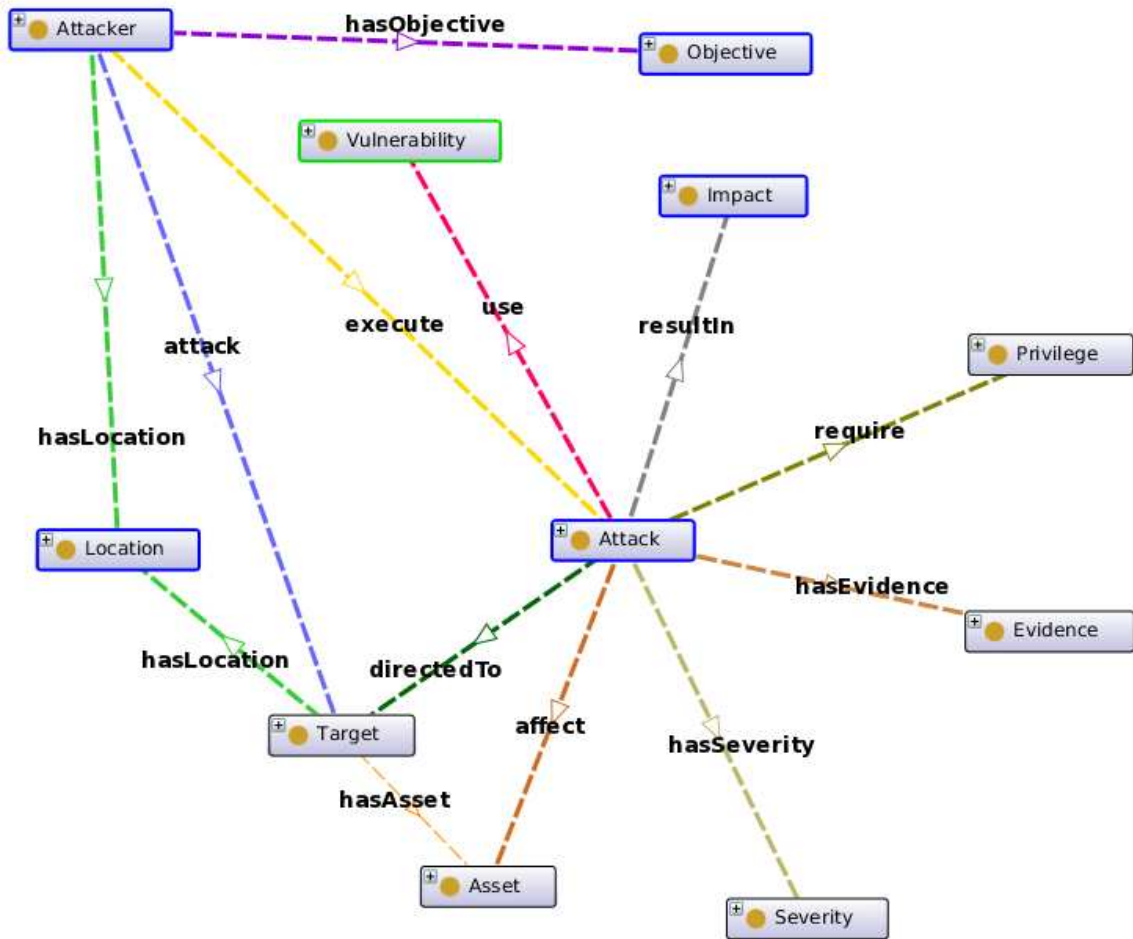


Figure 4.5: Ontology Upper Level Classes Snapshot

logical relations. A relation in the ontology can have one or more of the following characteristics:

- **Inversive:** any relation could have a corresponding inverse relation. If a relation r_j is an inverse relation of r_i , then the domain concept of r_j is the range concept of r_i and the range concept of r_j is the domain concept of r_i . For instance, the relation **attackedBy** is an inverse relation of the relation **attack**.
- **Transitive:** let us assume that relation r links class a to class b , and class b to class c . Relation r is a transitive relation if we can infer that r links a to c . An example of transitive relation in our ontology is the **subclass-Of** relation.

Based on the transitivity of the characteristic of the the **subclass-Of** relation, a reasoning engine can infer, for instance, that **Botnet** is a subclass of **Attack** (see Figure 4.2).

- **Symmetric:** If relation r is a symmetric relation and r links entity x to entity y , then we can infer that r links entity y to entity x . An example, of symmetric relation in our intrusion ontology is the **isRelevanceAlert** relation that connects an alert instance to another alert instance.
- **Asymmetric:** If relation r is an asymmetric (anti-symmetric) relation and r links individual x to individual y , then y cannot be linked to x via r . An example of asymmetric relation is the **nextStage** that connects an attack stage instance to another attack stage instance (see Figure 4.7).
- **Reflexive:** If relation r is a reflexive relation, then r must link individual x to itself. An example of reflexive relation in our intrusion ontology is the **isSimilarTo** relation that links an alert instance x to another alert instance y . This means that alert instance x is at least similar to itself.
- **Irreflexive:** If relation r is an irreflexive relation, and r links an instance x to instance y , then we can infer that x and y are not the same¹. An example of irreflexive relation in our ontology is the **attack** relation. If x attack y then x and y are not the same, because an attacker does not attack himself.
- **Functional:** If relation r is a functional relation, then there can be at most one individual y linked to an individual x via r . For example, if r is functional and individual x is related to individuals y and z , then y and z are the same (refer to the same individual).

¹In ontology unique names are not sufficient to distinguish between instances (individuals). The names Alice and Bob might refer to the same individual unless we explicitly state that they refer to different individuals.

Binary relation is the most common type of relations in ontology. However, heavy-weight method and task ontologies mostly require the use of N-ary relations. For instance, the representation of the following knowledge: "*Apache web server is a target of denial of service attack with high confidence and high severity*" requires the use of N-ary relations and cannot be represented by two binary relations because the relations in this case are all interconnected. To represent specific ontological relations (N-ary relations) we use two patterns. The first pattern is by creating a new class with N properties to represent the N-ary relation. The second pattern is by using lists of arguments to identify a N-ary relation that represents a sequence of arguments.

For instance, we represent the N-ary relation in the above statement using the first pattern by defining a relation called *attack diagnosis*, which describes the relation between *asset* (*Apache web server*), *attack* (*DOS*), *attack confidence*, and *attack severity*. In our ontology we create a new class to represent this relation as depicted in Figure 4.6.

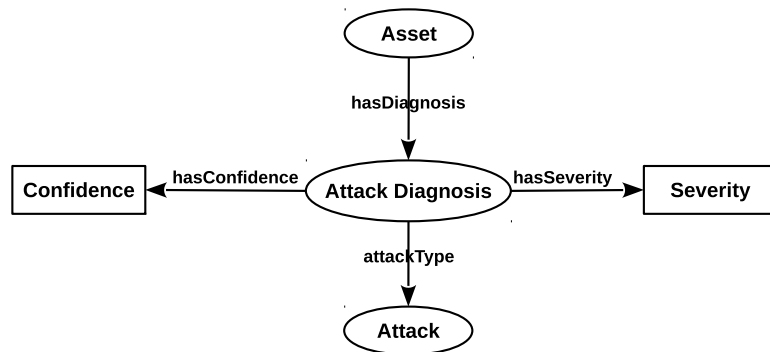


Figure 4.6: Attack Diagnosis Relation

Another intrusion analysis task that requires N-ary relations is when we attempt to describe the scenario of a multistage intrusion. A multistage intrusion scenario is a collection of attacks executed in specific order. To describe this case we need to use the second N-ary relation representation pattern mentioned above. To do that we will define two classes named *MultistageIntrusion* and *AttackStage*. The

MultistageIntrusion class encapsulates a scenario that consists of a collection of *AttackStage* instances. The *AttackStage* class is linked to itself via the *nextStage* and *previousStage* relations, which are used to describe sequence of attacks in the scenario. In addition, the *AttackStage* class is linked to an attack instance via the *hasAttack* relation, used to describe the attack executed in each stage. To determine the beginning and end of the attack scenario, we define the classes *FirstAttackStage* and *FinalAttackStage* as subclasses of *AttackStage* with an additional restriction that set the value of the *nextStage* link in *FinalAttackStage* to null and the value of the *previousStage* link in *FirstAttackStage* to null. Figure 4.7 illustrates the attack scenario relation.

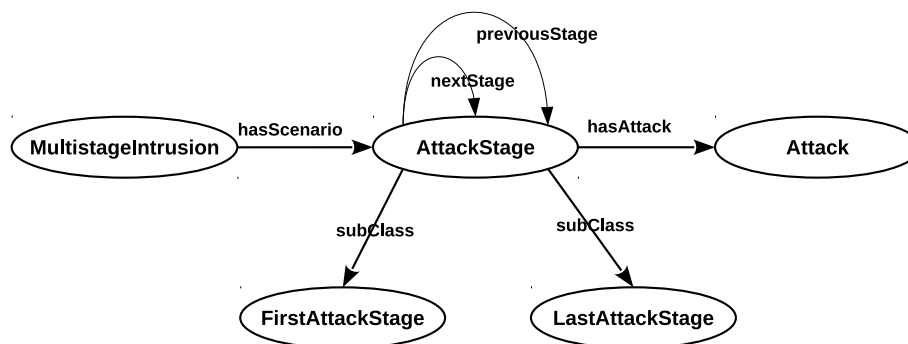


Figure 4.7: Attack Scenario Relation

The last step in this activity is to create the ontological relations dictionary. The relations dictionary lists the names of the relations, the domain, the range, and other characteristics. Table 4.2 shows an example of an entry in the relations dictionary.

Attribute	Value
Relation Name	hasAttack
Domain	Attacker
Range	Target
Properties	Inversive, Irreflexive
Description	Relate the source and the destination of an attack.

Table 4.2: Example of Entry in the Relations Dictionary

Class Semantic Description

The next activity in the conceptualization stage is to formally describe the classes in our ontology. To this point we only created a collection of named classes where each class has a name which is a set of characters. This is only useful for humans, but a machine cannot interpret or understand the meaning of these names without a formal description. In other words, an inference engine cannot really infer anything using solely classes names. Class formalization requires defining the restrictions on the properties of the classes. In general, there are three types of property restrictions as follow:

- **Quantifier Restrictions:** this type of restrictions is divided into two subtypes, namely, existential restrictions and universal restrictions, which are denoted by the symbols \exists and \forall , respectively.
- **Cardinality Restrictions:** this type of restriction is used to describe a class, which has at least, or at most, or exactly a specified number of relations with other classes or data.
- **Value Restrictions:** a value restriction is used to specify that a class has a relation with a specific instance/individual of another class.

The above different types of restrictions allows describing formally the classes in the ontology by defining necessary and sufficient conditions. A class described with necessary conditions is a primitive class. On the other hand a class described with both necessary and sufficient conditions is a defined class. Based on the intended use of the ontology, we can decide which class is going to be primitive and which one is going to be defined. But in general, the top level or abstract classes are usually primitive classes and the low level or more concrete classes are defined classes.

To explain the difference between the use of necessary conditions on one hand and the use of necessary and sufficient conditions (primitive and defined classes) on the other hand, and how they affect the inference process, we will use in the following a simple example consisting of providing the semantic description of the class *Attack*. Simply, we will say that an *Attack* must have at least an *Impact* and at least affect an *Asset*. The formal description of class *Attack* can be constructed using existential restriction as follows:

$$\text{Attack} \sqsubseteq \left[\begin{array}{l} \exists \text{ hasImpact.Impact} \\ \exists \text{ affect.Asset} \end{array} \right]$$

The above semantic description uses only necessary conditions to formally express the fact that for some activity to be an attack it is necessary for it to be in relationship with an instance of the class *Impact* via the relation *hasImpact* and to be in relationship with an instance of the class *Asset* via the relation *affect*. However, this does not mean if an individual x is in relationship with an individual y , which is a member of the class *Impact* via the relation *hasImpact* and with an individual z , which is a member of the class *Asset* via the relation *affect*, that an individual x is a member of the class *Attack*. The inference engine will not infer that x is a member of *Attack*. This is because the class *Attack* is described with necessary conditions.

On the other hand using necessary and sufficient conditions to define a class C allows inferring that if an individual x fulfills these conditions then it must be a member of the class C . For example, the class *DOSAttack* is described below using necessary and sufficient conditions, as an attack that at least has an impact *LossOfAvailability*:

$$\text{DOSAttack} \doteq \left[\begin{array}{l} \text{Attack} \\ \bigcap \exists \text{ hasImpact.LossOfAvailability} \end{array} \right]$$

The above description of the class *DOSAttack* means that if x is a member of the *DOSAttack* class then x must be a subclass of *Attack* and have a relationship with an individual y that is a member of the class *LossOFAvailability* via the relation *hasImpact*. It also means if an individual x is subclass of *Attack* and has a relationship with an individual y that is a member of the class *LossOFAvailability* via the relation *hasImpact*, then x must be a member of the class *DOSAttack*.

Of course, necessary and sufficient conditions are not limited to existential restrictions. In most cases we use existential, universal, and cardinality restrictions. For example, we could describe a MultiImpactWebAttack as follows:

$$\begin{aligned} \text{MultiImpactWebAttack} \doteq & \text{[Attack} \\ & \cap \exists \text{ min}(2) \text{ hasImpact.Impact} \\ & \cap \exists \text{ affect.WebServer} \\ & \cap \forall \text{ affect.WebServer]} \end{aligned}$$

It is important to document the classes and their ontological characteristics. This is performed by creating the class dictionary. Table 4.3 depicts an example of entry in the class dictionary.

Attribute	Value
Class Name	DOSAttack
Type	Defined
Domain Relations	result-In, hasMitigation, affect
Range Relations	execute, report, victim-Of
Description	an attack that results in loss of the availability of the target or one of its assets

Table 4.3: Example of Entry in the Class Dictionary

The final activity in the conceptualization stage is to define the inference rules. The inference rules play an important role in extracting implicit knowledge from the ontology. The inference engine processes the inference rules to identify new or

implicit knowledge from asserted and explicit knowledge. Managing and maintaining inference rules is a key challenge in ontology engineering. Therefore, it is important to define only the inference rules which support the intended use of the ontology.

To define the inference rules required for intrusion alerts analysis and attack intelligence extraction, we begin by informally describing the rules using natural language (NL) and make sure that the rules satisfy the ontology specification, e.g., the rules should help in answering the competency questions. Then, for each inference rule we identify the ontology classes and relations required to describe the rule. It is possible at this point that we will need to define new relations and classes to satisfy the description of the inference rules. Finally, we need to identify the variables for each rule. Each inference rule consists of two parts, namely, the antecedent or the body of the rule, and the consequence or head of the rule. Whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

4.3.3 Formalization

The third stage in the ontology development lifecycle is the formalization stage, where the ontology is implemented using an ontology language. The selection of the ontology language depends on the ontology specification and the intended use of the ontology. In our case we have a heavyweight method ontology, therefore we can either use frames or description logic languages. As mentioned before, to carry out the formalization, we use description logic as our ontology knowledge representation paradigm.

Description logic (DL) is a family of knowledge representation languages. We use description logic to describe the main components of the ontology. The knowledge base in description logic is divided into two parts, the TBox and the ABox. The TBox contains terminological knowledge consisting of the definition of concepts and relations. The Abox contains the assertional knowledge consisting of the definitions

of instances. We use in this work the Web Ontology Language (OWL) as our ontology encoding language. OWL is a description logic based language for developing ontologies and representing knowledge in semantic web. To describe our inference rules we use the Semantic Web Rule Language (SWRL).

The following is an example of an inference rule that finds if two alerts have the same attacker:

$$\text{Alert}(?x) \wedge \text{Alert}(?y) \wedge \text{Attacker}(?a) \wedge \text{hasSource}(?x, ?a) \wedge \text{hasSource}(?y, ?a) \rightarrow \text{hasSameAttacker}(?x, ?y)$$

The above inference rule is encoded using SWRL and stored as XML files in the knowledge base. In SWRL, all rules are expressed in terms of OWL concepts (classes, properties, individuals, literals, etc). Table 4.4 shows some of the predicate sentences (used to define the rules) and their meanings.

Predicate Sentence	Description
Alert(?x)	check if variable x is an Alert instance
Attack(?a)	check if variable a is an Attack instance
report(?x,?a)	check if variable a which is an attack instance is reported by x which is an alert instance
Impact(?m) \wedge resultIn(?a,?m)	check if variable a which is an attack instance has an impact m which is an instance of attack impact class

Table 4.4: Predicate Examples

A chain of rules can be used to infer an indirect relation between two alerts. For example, it can be established by inference that two different alerts that report two different attack types while having the same impact are relevant. An example of SWRL rule to infer alerts with similar attack impact is given by:

$$\begin{aligned}
& Attack(?a) \wedge Attack(?b) \wedge Impact(?m) \wedge resultIn(?a, ?m) \wedge resultIn(?b, ?m) \rightarrow \\
& \quad hasSameImpact(?a, ?b) \\
& Alert(?x) \wedge Alert(?y) \wedge Attack(?a) \wedge Attack(?b) \wedge report(?x, ?a) \wedge report(?y, ?b) \wedge \\
& \quad hasSameImpact(?a, ?b) \rightarrow reportSameImpact(?x, ?y)
\end{aligned}$$

4.3.4 Validation

The last stage in our ontology development lifecycle is the validation stage. The goal of this stage is to make sure that the developed ontology satisfies its intended use. Usually, the ontology validation can be accomplished using one or more of the following methods:

- *Competency Checking*: if we can answer all the competency questions then we can claim that the ontology contains sufficient components (classes, relations, etc) to represent the domain of interest with respect to the competency questions. Hence, we can think of the competency questions as kind of requirements specification for the ontology.
- *Tell and Ask*: the tell and ask method simply consists of designing a set of logical assertions (tell) and then issuing queries (ask). Then, the answers provided by the ontology to the queries are evaluated against the logical assertions.
- *Field Testing*: consist of testing the developed ontology in the target application environment and observing how the ontology performs with respect to its intended use.

It is important to point out that the process of ontology development is an iterative process. This means we might find at the end of the validation that we need to add new classes, relations, or restrictions to the ontology or redo a stage in the development lifecycle.

4.4 Reasoning with Ontology

Ontology in itself is only a sophisticated knowledge representation approach. Likewise, we still need a reasoning system or an inference engine that can make use of the knowledge encoded in the ontology. Reasoning over ontology is the process of finding implicit facts given explicitly stated facts in the ontology. Ontology reasoning is useful for generalization, prediction, diagnosis, and drawing conclusions from facts. In general there are three main forms of reasoning that can be implemented over ontologies, namely, deductive reasoning, inductive reasoning, and abductive reasoning.

4.4.1 Deductive Reasoning

Deductive reasoning is the most common reasoning approach over ontologies. Deductive reasoning is used to draw a conclusion by narrowing down general domain knowledge encoded in the ontology. A key property of deductive reasoning is that the inferred result is guaranteed to be true as long as the assertions used in the inference process are true. For instance, let us consider the following premises: *P1: all DOS attacks result in loss of availability* and *P2: Land is a DOS attack*. Given the above premises, using deductive reasoning, we can conclude that: *Land attack results in loss of availability*.

To our knowledge all the existing inference engines for DL-based ontologies support only deductive reasoning. Using deductive reasoning and the available inference engine for DL-based ontology, a reasoner can perform the following tasks:

- Calculate Entailment: using the asserted knowledge (explicit facts and axioms) the reasoner infers logical consequences. In this case the inferred knowledge (implicit) is a logical consequence of the premises.
- Calculate Subsumer: given a set of classes the reasoner discovers the least com-

mon subsumer (LCS), which is the class that subsumes the given set of classes.

- Instance Realization: given an instance x and assertions about x the reasoner finds the classes that x belongs to.
- Instance Checking: given a class C and an instance x the reasoner decides whether x is an instance of C or not.
- Consistency Checking: the reasoner checks the ontology to detect any contradictory axioms. An ontology O is consistent if there is at least one model M that satisfies the facts F in O .
- Conjunctive Query Answering: given an ontology O and a conjunctive query q , the reasoner returns the answer of q with respect to O

4.4.2 Inductive Reasoning

Inductive reasoning is a bottom-up reasoning approach that is based on observing instances, recognizing patterns and making generalizations based on those patterns. An important difference between inductive reasoning and deductive reasoning is that in inductive reasoning the truth of the premises does not guarantee the truth of the conclusion. For example, let us assume that we have n number of intrusions and all of these intrusions are *instance-of* Privilege Escalation attack. In addition, each of these instances *Affects* a FTP server and has the impact of root privileges. Given these facts and using inductive reasoning we can conclude that all privilege escalation attacks that target FTP servers will allow the intruder to gain root privileges.

4.4.3 Abductive Reasoning

Abductive reasoning aims at finding the best explanation for an observed case or fact. Specifically, abduction reasoning allows the precondition \mathbf{a} to be inferred from the consequence \mathbf{b} . For instance, let us consider three intrusion instances \mathbf{A} , \mathbf{B} , and \mathbf{C} , respectively, executed in sequence by the same intruder. Let us also assume that from the ontology we know that the intrusion instance \mathbf{A} is an FTP probing attack and that the intrusion instance \mathbf{C} is based on an FTP exploit that requires the intruder to have root privileges. Given these facts we can infer that the intrusion instance \mathbf{B} is a privilege escalation attack that has the impact of root privileges.

4.5 Semantic Analysis and Correlation

The word *Semantics* means the study of meanings ². Semantics analysis refers to the process of extracting and representing knowledge about classes. In the discipline of artificial intelligence, semantics analysis aims at representing knowledge that is understandable by machines, so that they can perform tasks that require human intelligence. In fact, the use of semantics analysis can enable software programs to intelligently reason about their content and goals.

The idea of using semantic analysis to design intelligent systems dates back to the late 1990s. However, the emergence of semantic web in the last decade made the development of sophisticated semantic-based intelligent systems possible. Perhaps today the most notable use of semantics analysis is in the semantics web, biomedical applications, and e-commerce applications [20, 54, 9, 87].

As we mentioned in Chapter 3, intrusion alerts consist of complex and symbolic data. In our opinion using the semantic analysis to investigate IDS alerts will improve

²Merriam-Webster's Collegiate Dictionary

the intrusion alert analysis process. The first step to apply semantic analysis is to create semantic description. In other word we need to provide semantic description for every concept and individual in the intrusion analysis domain to be able to use semantic analysis in IDS alert analysis. This first step was accomplished by the creation of our proposed intrusion analysis ontology.

In general, alerts correlation consists of relating any number of alerts with some identifiable patterns. A set of alerts are correlated based on the relationships occurring between them. These relations could be captured based on statistical measurements, a set of rules or some similarity measurements. In our work, alerts are correlated based on their semantics relationships. There are three forms of semantic correlations [50]. The first form is based on the semantic equivalence between classes, attributes and instances. The second form of semantic correlation is based on the taxonomic relations (e.g. inheritance relation) that exist between classes. The third form of semantic correlation is based on the ontological relations that exist between the components of the ontology (classes, attributes, and individuals).

Semantic-based alerts correlation is the process of correlating alerts using the semantic of their attributes with respect to the taxonomic and ontological relations that occur between their attributes in a given intrusion ontology. To build an effective semantic-based alert analysis or correlation system, it is important to be able to measure the semantic similarity and the semantic relevance between IDS alerts.

4.5.1 Ontology-based Semantic Similarity

Semantic similarity provides an objective measure of the closeness between two or more concepts based on the similarity of their meanings or semantic descriptions.

Similar concepts or classes in an ontology are structured in a taxonomy structure also referred to as *concept tree*. A *concept tree* describes the abstraction relation-

ship (i.e. generalization/specialization) between similar concepts using a hierarchical structure. The root of the tree corresponds to the most abstract form of the concept, while intermediary nodes correspond to refined concepts, and leaf nodes correspond to instances.

The similarity between two concepts in an ontology depends on the commonalities and the differences between the two concepts. The commonalities between two concepts are represented by their relations to their lowest common ancestor in the ontology. On the other hand the differences between them is based on their locations within the ontology structure. Based on the above considerations, given two concepts a and b we define our semantic similarity metric between a and b as follows:

$$sim(a, b) = 1 - \frac{(path(a, LCA(a, b)) + path(b, LCA(a, b)))}{(depth(a) + depth(b))} \quad (4.5.1)$$

Where $path(a, LCA(a, b))$ is the length of the shortest path from concept a to the least common ancestor (LCA) of a and b in the concept tree, and $depth(a)$ is the depth of concept a in the concept tree. The proposed similarity measure is a number between $[0, 1]$ where 1 corresponds to exact match and 0 corresponds to no match between the concepts.

The metric has two important properties. The first property is that the semantic similarity between higher-level concepts are less than the semantic similarity between lower-level concepts. This reflects the fact that two general concepts are less similar than two specialized ones. The second property is that the semantic similarity between a parent concept and any child concept of this parent is greater than the similarity between this child concept and any other child concept of the same parent.

To illustrate the use of semantic similarity in IDS alert analysis, we will use a simple example to measure the similarity between different classes of attacks. Let

us assume we have a concept tree that represents the *information gathering* attack. Figure 4.8 is a subtree that describes part of *Information Gathering* attack taxonomy.

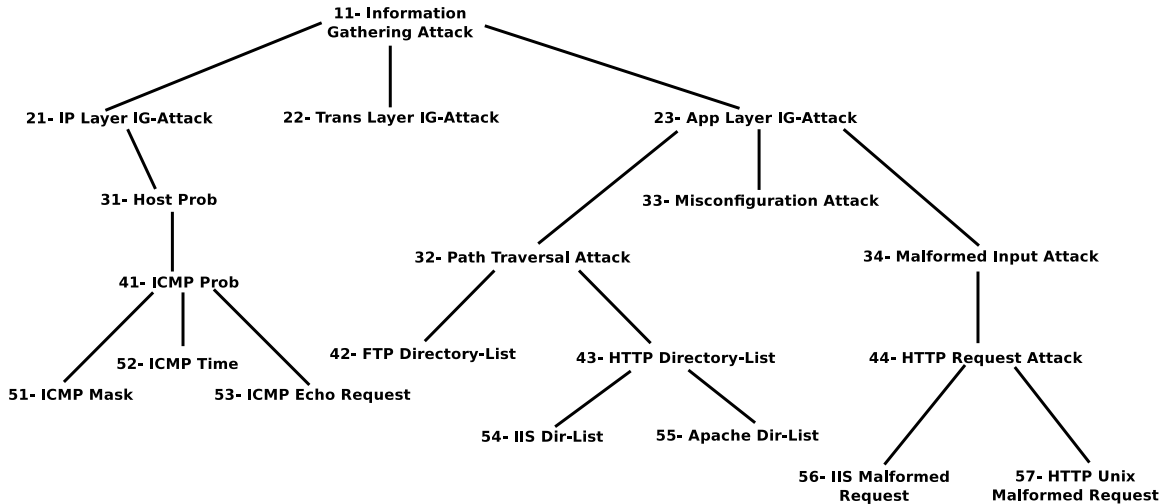


Figure 4.8: Information-Gathering Attack Ontology (Partial)

Given the concept tree, the computation of the semantic similarity between any two concepts in the ontology is straightforward. For instance, using equation 4.5.1, the semantic similarity between the IIS Dir-List class and the Apache Dir-List is computed as $sim(IISDir - List, ApacheDir - List) = 0.8$. In this example, the class HTTP Directory-List is the first common ancestor of IIS Dir-List and Apache Dir-List, and the depth of IIS Dir-List and Apache Dir-List equal 5.

We use the notion of concept tree to measure the similarity between symbolic alert attribute values as explained in the following. Let a_1, \dots, a_n denote a set of IDS alerts, where each alert a_i is represented using a p -dimensional attribute vector $[a_{i1}, \dots, a_{ip}]$ and only the first s attributes are symbolic attributes ($1 \leq s \leq p$).

Given two alerts $a_i = [a_{i1}, \dots, a_{ip}]$ and $a_j = [a_{j1}, \dots, a_{jp}]$, the semantic similarity between symbolic attribute values a_{ik} and a_{jk} ($1 \leq k \leq s$) can be calculated using equation 4.5.1 as following:

$$sim(a_{ik}, a_{jk}) = 1 - \frac{(path(a_{ik}, LCA(a_{ik}, a_{jk})) + path(a_{jk}, LCA(a_{ik}, a_{jk})))}{(depth(a_{ik}) + depth(a_{jk}))} \quad (4.5.2)$$

We define the semantic similarity between two alerts a_i and a_j as follows.

$$sim(a_i, a_j) = \frac{\sum_{k=1}^s sim(a_{ik}, a_{jk})}{s} \quad (4.5.3)$$

The semantic similarity between any pair of alerts or between any pair of alert attributes is a value between **0** and **1**. Where **1** indicates the maximum similarity and **0** indicates that there is no similarity at all between the two attributes or alerts.

The computation of semantic similarity based on the ontology structure has two advantages. First, we can easily measure the similarity between symbolic data (e.g. classes and complex concepts) in the intrusion analysis domain. Second, previous techniques in alert analysis define the similarity between alerts using ad-hoc methods. In contrast, ontology-based semantic similarity provides a systematic method to measure similarity between alerts and other concepts in the intrusion analysis domain. Now, if we add, remove, or modify any concept in the ontology we can easily calculate the similarity between this concept and other concepts in the ontology.

4.5.2 Ontology-based Semantic Relevance

In knowledge engineering and information retrieval, the notion of *relevance* expresses how two objects are related with respect to the matter at hand. Semantic relevance occurs between classes and individuals in the same ontology either by explicit relations or implicit relations. Several approaches have been proposed to calculate the semantic

relevance between concepts, objects or resources in specific domain of knowledge [63, 62].

The notion of semantic relevance might seem equal to the notion of semantic similarity, however, they are different. Semantic similarity between two concepts can be measured based on the similarity of their contents or based on the taxonomic relations (e.g is-A, subclass, instance-of) between them in a given ontology. On the other hand two concepts can be semantically relevant as long as there is an explicit or implicit relation between them even if their contents are completely different and they are not connected by any taxonomic relation.

Using semantic relevance we can measure relatedness between a group of concepts or a group of instances in the ontology. For example, we can measure how two or more alerts are relevant to each other or how two or more attacks are relevant to each other. Calculating the semantic relevance between concepts or instances depends on the relations that exist between these concepts or instances. For example, let x and y be two IDS alert instances, where alert x reports a scan attack against a web server and alert y reports a buffer overflow attack against a database server. Now, x and y are semantically similar because both are IDS alerts. But, whether x and y are semantically relevant will depend on the ontological relations that exist between them. If x and y are connected by ontological relations then they can be considered as semantically relevant.

The ontological relations could be explicit or implicit. Based on explicit relationships and inference rules, semantic inference can be used to discover the implicit relationships between intrusion instances, alerts, and other alerts attributes. Both the explicit and (discovered) implicit relations are used to calculate the semantic relevance. A subset of the ontological relations used to calculate the semantic relevance between alerts are shown in Figure 4.9.

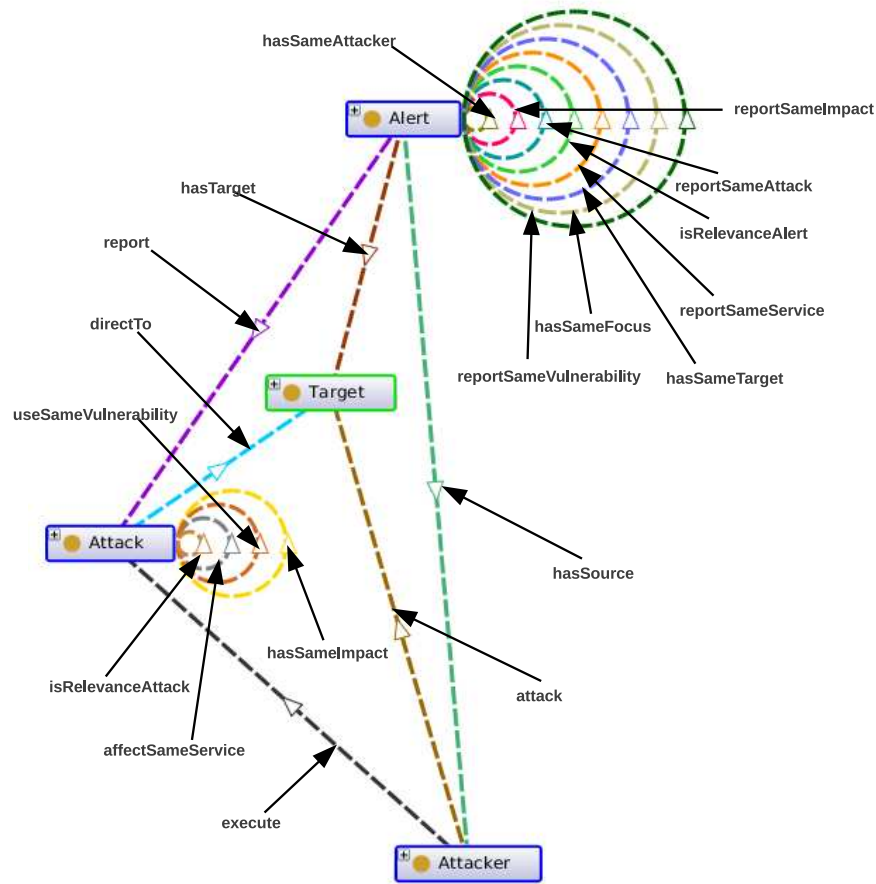


Figure 4.9: Ontological Relations between Alerts, Attack, Attacker and Target

In Figure 4.9 the explicit relations have different domain and range (e.g. $\text{report}(\text{Alert}, \text{Attack})$), while implicit relations have the same domain and range (e.g. $\text{hasSameSource}(\text{Alert}, \text{Alert})$).

The semantic relevance between two alerts is based on the relations occurring between them. Hence, we compute the semantic relevance between two alerts x and y as the summation of the weights of all the relations occurring between them divided by the summation of the weights of all the relations that can occur between any two alerts.

Given two alerts $x \in A, y \in A$, let R_{xy} denote the set of all relations between x

and y . Let R denote the set of all relations between two alerts from A , i.e., $R = \cup_{x \in A, y \in A} R_{xy}$. Given a relation $r \in R$, let $w(r)$ denote the weight associated with r . We define the semantic relevance between alerts x and y as follows:

$$sem_{rel}(x, y) = \frac{\sum_{r \in R_{xy}} w(r)}{cardinality(R)} \quad (4.5.4)$$

It is important to understand that the semantic or the interpretation of the inferred relations (e.g. *hasSameSource*, *reportSameAttack*, *useSameVulnerability*, etc.) is not limited to equality or perfect match. For example, two IDS alerts could be related to each other by *hasSameSource* as long as their sources belong to the same subnet, domain or even the same geographical location. The same applies for *reportSameAttack*, where the attacks reported by the alerts might appear different (e.g have different signatures), however, they affect the same system, result in the same impact, etc. In fact modifying the the semantics of the ontological relations can allow designing different alerts analysis and attack intelligence extraction.

In, addition the semantic relevance between two concepts c_i and c_j can be inferred even if there is no direct ontological relation between them. For example, c_i has an ancestor concept that shares an ontological relation with another concept that is an ancestor of concept c_j . In this case we can say that c_i and c_j are semantically relevant to each other through inheritance.

4.6 Summary

In this chapter we introduced a novel intrusion ontology. The proposed ontology is a heavyweight ontology in terms of formality and granularity. In addition, our intrusion ontology is a method ontology that contains problem solving knowledge.

We explained the ontology engineering process to build our intrusion ontology. The proposed ontological engineering process can be used to build method ontologies for other domains such as network forensic, vulnerability analysis, etc. We also introduced the use of semantic analysis and semantic correlation to analyze IDS alerts and explained how the semantic correlation can take advantage of the proposed intrusion ontology to measure semantic similarity and semantic relevance. Our ontology-based semantic similarity metric has several appealing characteristics over other semantic similarity metrics proposed in the literature.

Chapter 5

Novel Alert Analysis Techniques

As we mentioned in Chapter 4, the intrusion analysis ontology provides the knowledge required for IDS alerts analysis. However, the ontology by itself cannot perform any of the alert analysis tasks. Therefore, it is important to design techniques that can use the knowledge encoded in the ontology to perform these tasks.

In this chapter, we propose new algorithms for IDS alert filtering/verification, IDS alert aggregation/summarization, predicting missing attack (false negative), detecting attack patterns and reconstructing attack scenarios. The techniques we developed in our work use machine learning and benefit from the rich knowledge structure (classes and their relations) from the intrusion ontology.

5.1 Target Network Example

To explain the proposed alert analysis techniques we will use a simple example of target network in a small e-commerce company. This target network receives both normal and malicious traffic. An abstract network topology of the target network is illustrated by Figure 5.1.

As illustrated in Figure 5.1, the target network consists of three subnets. The first

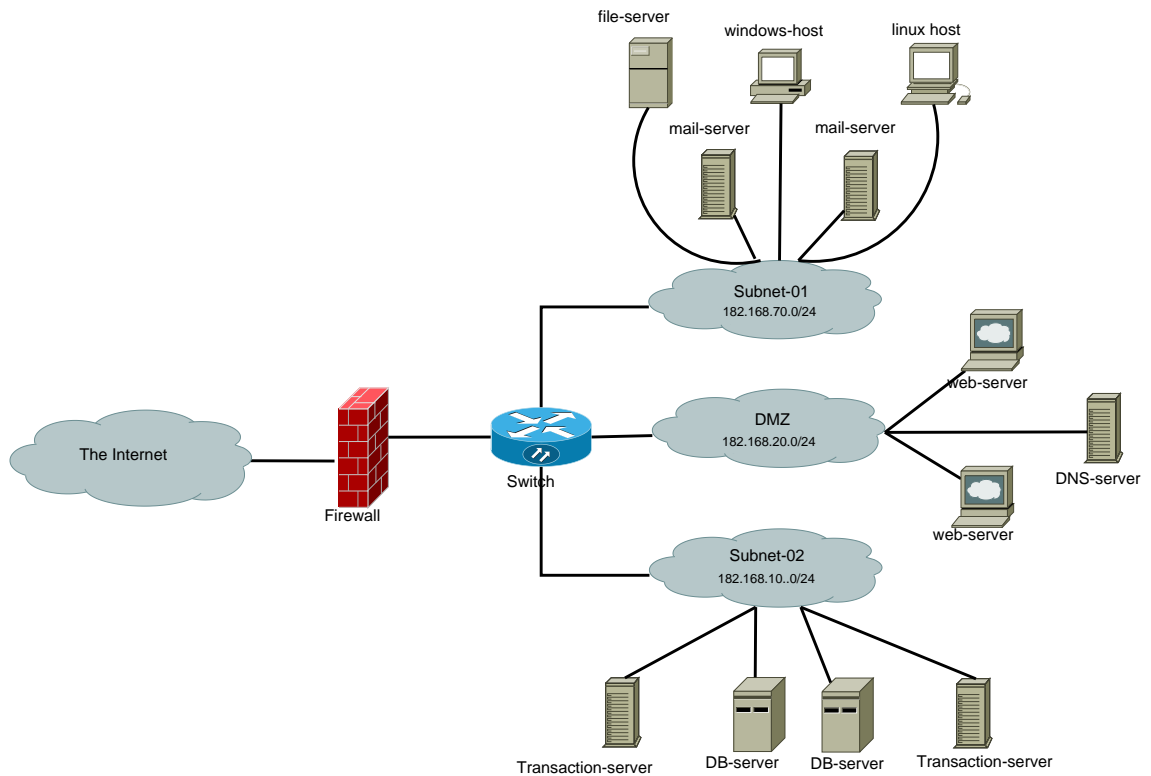


Figure 5.1: Target Network Topology

subnet (labeled subnet-01) contains file servers, mail servers and hosts used by the company employees. The database and transaction servers are not accessible from the Internet; they can only be accessed by the web servers in the DMZ and the machines in "subnet-02". The customers' purchases and orders are processed and stored in database servers located in a secured subnet (labeled subnet-02). The third subnet (labeled DMZ) is a DMZ (demilitarized zone), in which there are two web servers and a domain name server; these servers can be accessed by the public (e.g. from the Internet). The customers can access the web server, and purchase company products. Table 5.1 lists the characteristics of the hosts in the target network including their IP addresses, subnets, operating systems, running services and roles in the network. As we can see, the target network consists of heterogeneous platforms, including hosts OS and services.

Now, using the ontology we will extract several concept trees describing the taxonomic relations between the key classes (concepts) in the target network. Here, in our example we will use simple concept trees to explain our IDS alert analysis techniques. We will assume that our ontology contains the following key concepts: Attack, Asset, Address, and Time.

Network	Machine IP	Platform	Service	Role
DMZ	192.168.20.70	Windows XP	MS IIS v6.0	Web Server
	192.168.20.80	Windows 7	MS IIS v7.5	Web Server
	192.168.20.90	Windows 2008	Bind 8.2	DNS Server
Subnet-01	192.168.70.50	Windows 7	Exchange 2010	Mail Server
	192.168.70.20	Fedora 10	Sendmail 8.14.4	Mail Server
	192.168.70.33	Ubuntu 10.4	SSH	Host
	192.168.70.31	Ubuntu 11.4	SSH	Host
	192.168.70.22	Windows XP	RDP	Host
	192.168.70.25	Windows 7	RDP	Host
	192.168.70.21	Windows 7	RDP	Host
192.168.70.30	Windows XP	FileZilla 0.9.31	File Server	
Subnet-02	92.168.10.17	Windows 7	MS SQL 2005	DB Server
	192.168.10.12	Windows 2000	MySQL 4.5.2	DB Server
	192.168.10.84	FreeBSD	SSH, RSH	Transaction Server
	192.168.10.88	FreeBSD	SSH, RSH	Transaction Server

Table 5.1: Description of the hosts in the target network

Figure 5.2 shows an example for the attack concept tree. The leaf nodes in the tree represent the attack signature-id. IDS tools use signature-id in the alerts to represent the reported attack instances.

Figure 5.3 shows an example for the address concept tree. The address concept tree illustrates how we understand the similarity between IP addresses. The tree consists of two main subtrees, one for internal addresses and the other for external addresses. The first subtree organizes the target network IP addresses based on their location in the network topology and their roles in the network. The second subtree organizes the external IP addresses based on their geographical locations. Figures 5.4

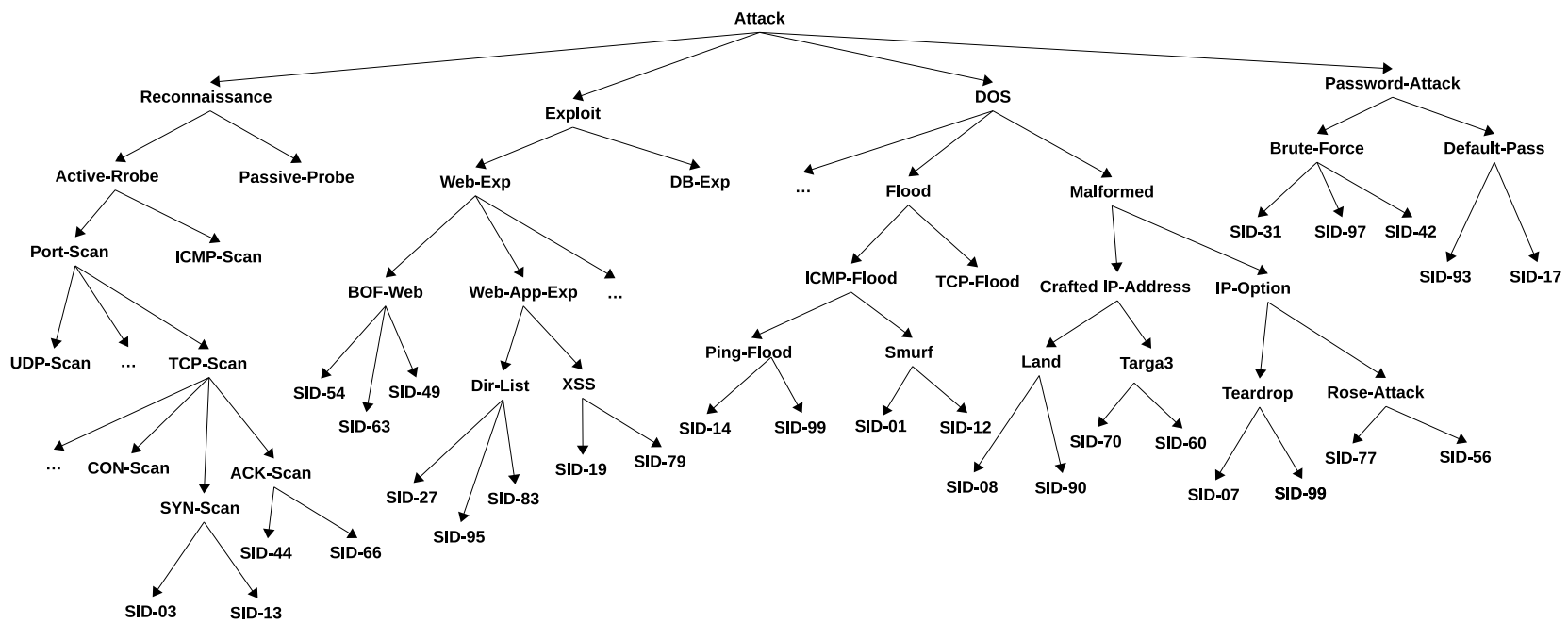


Figure 5.2: Attack Concept Tree Example

and 5.5 show examples for the assets and the time concept trees, respectively.

5.2 IDS Alert Verification

IDS alerts verification is a classification problem. We try to classify alerts either as true positive or false positive. The challenge when we deal with IDS alerts is how do we represent the alerts features that are suitable for building a robust classifier. Selecting the features or the attributes to represent alerts messages is the most important step for distinguishing between false and true alerts.

An alert message contains a set of attributes that vary from one IDS sensor to another. However, alert messages from different IDS sensors try to provide the same information. The most common alert message attributes are the attack, the source of the attack, the target of the attack, and the detection timestamp. The alert message itself does not contain sufficient information to distinguish between true and false alerts. In other words, the attributes are not suitable for building an alert classifier. The intrusion analyst interprets the alert message and translates the alert message attributes to more complex attributes that are useful for alerts classification.

In our alert verification approach we apply instance-based learning (example-based learning). Our approach begins with a set of training examples consisting of raw IDS alerts labeled as either true positive or false positive. An intrusion analyst can provide the labels for the training examples. Then, using the ontology we represent the context of each alert in the training example by calculating a set of features. Using the alert context we can classify new unlabeled alerts as true or false positive using one of two possible techniques. The first technique uses an instance-based learning algorithm such as k-nearest neighbors with semantic distance metric. The second technique uses inductive reasoning and a novel ontology-based rule induction

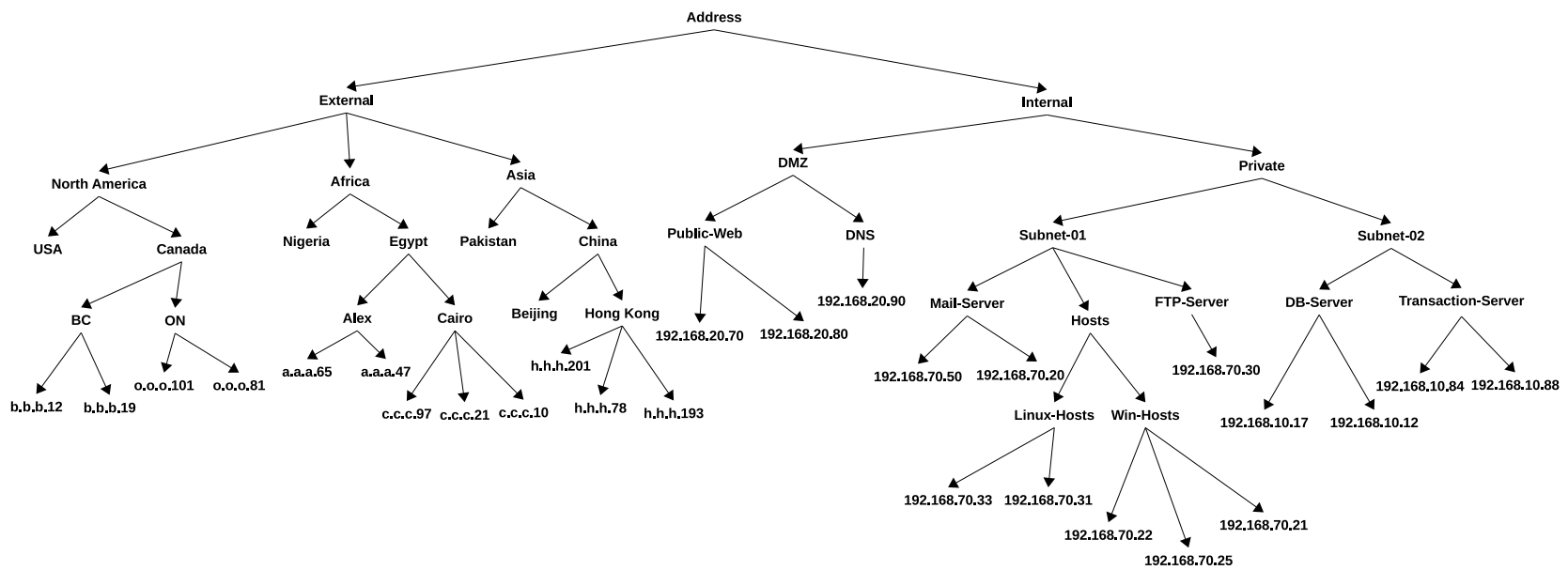


Figure 5.3: IP Addresses Concept Tree Example

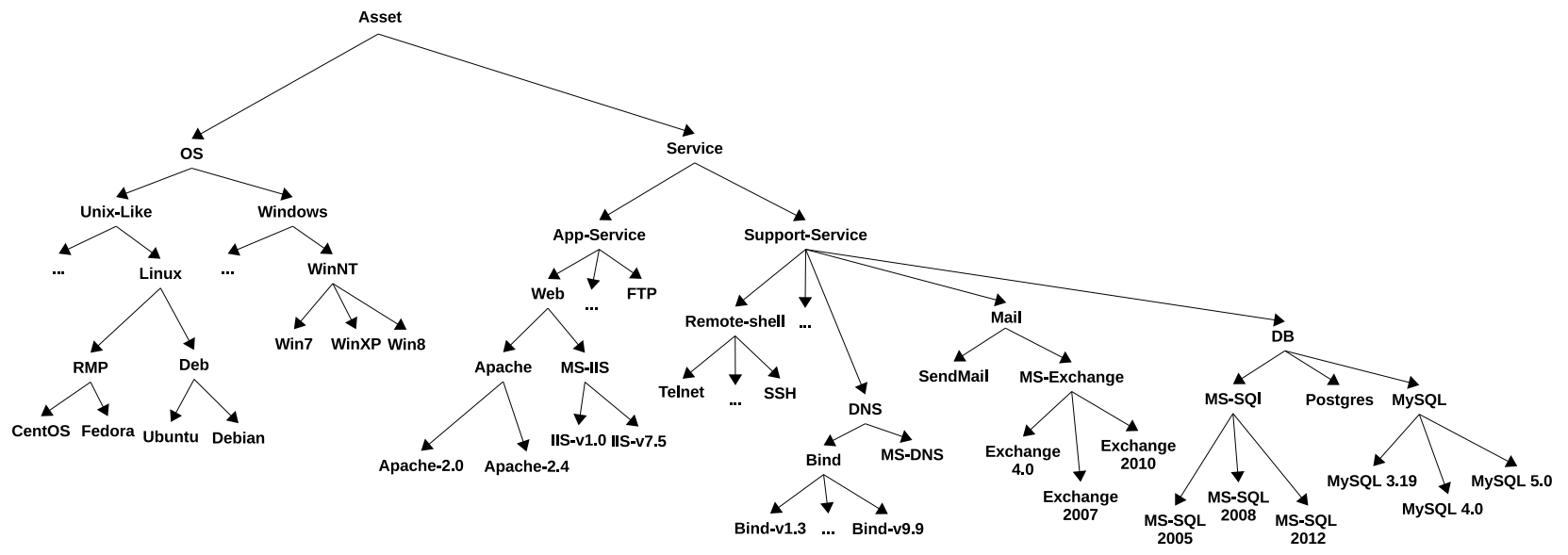


Figure 5.4: Asset Concept Tree Example

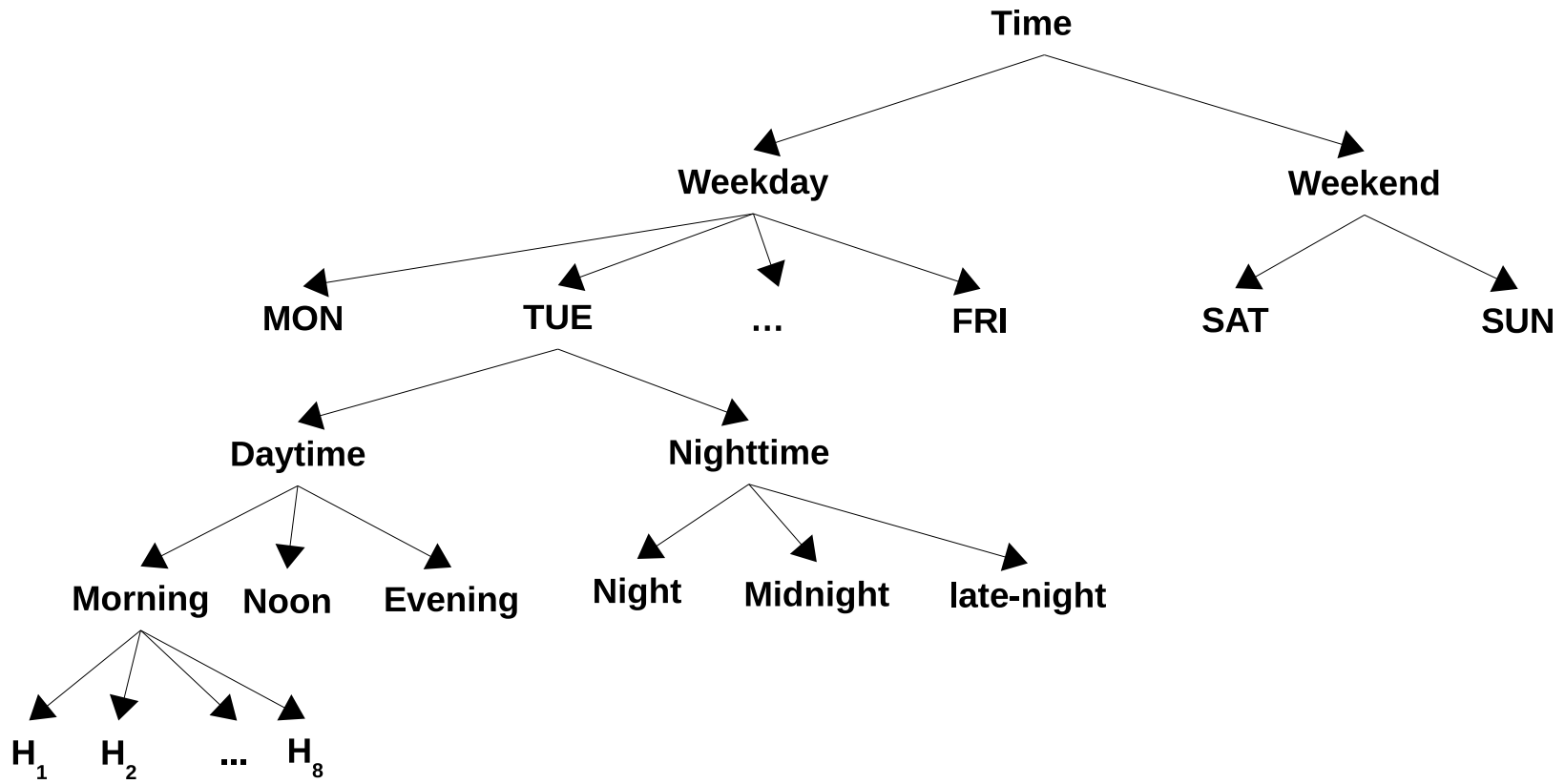


Figure 5.5: Time Concept Tree Example

algorithm introduced in this work to learn appropriate classification rules.

5.2.1 Alerts Context

Context is defined in Webster dictionary as "the interrelated conditions in which something exists or occurs". In other words, it is the circumstances that surround an event and can throw light on its meaning. Here, we try to distinguish between false and true alerts using their context.

When an intrusion analyst tries to decide if a given alert is either a false positive or true positive, he looks at the context of that alert. In other words, he thinks about the semantic of the attack reported by this alert, the semantic of the target, and the characteristics of the alerts stream (information about alerts that occur with the alert in-question). All of this information is used by the analyst to construct the context of the alert.

In our work each alert message (or shortly alert) consists of seven attributes, namely, the source IP address, source port, destination IP address, destination port, attack signature, protocol, and timestamp. Using the knowledge base we process the raw alerts and extract useful features that we can use to represent the alerts in a form useful for building a classifier.

Using the attributes in the raw alerts we can extract additional features from the knowledge base about the attack and target semantics to describe the alerts. We extract attack semantic features from the signature attribute associated with the raw alert. Attack semantic features describe the meaning of the attack. These include information such as the attack type, the required privileges to execute the attack and the services exploitable through the attack. This information can be obtained from the IDS documentation or any online vulnerability database such as Open Source Vulnerability Database (OSVDB) and Common Vulnerabilities and Exposures (CVE).

Table 5.2 shows some examples attack semantic features.

Feature Name	Possible Feature Values
<i>impact</i>	information gathering, code execution, etc
<i>vulnerability</i>	protocol flow, implementation bug, etc
<i>privilege</i>	any, local user, root, etc.
<i>services</i>	web, ftp, dns, mail, etc.
<i>platforms</i>	Windows, Mac, Linux, etc.
<i>protocols</i>	ICMP, TCP, UDP, etc.

Table 5.2: Attack Semantic Features

The target semantic information includes basic information about the target, e.g. the target operating system, running services and the access policy to the services, the general topology of the network like what IP addresses belong to the DMZ or private subnet, etc. Using the target IP address and port number attributes in the raw alerts, we can extract the above information from the knowledge base. Note that our technique does not require detailed information about the target network like configuration files, installed software, updated, patches, etc. Collecting this basic information can be easily achieved using a network reconnaissance or other network scanning tools. Table 5.3 shows some examples of target semantic features.

Feature Name	Possible Feature Values
<i>running service</i>	web, ftp, dns, mail, ssh, etc
<i>service access</i>	any, local, remote, remote-root, etc
<i>platform</i>	Windows, Linux, Solaris, etc.
<i>type</i>	Firewall, IDS, Server, Host, etc
<i>countermeasure</i>	firewall, IPS, etc

Table 5.3: Target Semantic Features

It is important here to explain that we are not using the network scanning tools to check or to verify that a vulnerability exists in the network or that an attack is successful in the target network.

The last group of features is extracted from the characteristics of the alerts stream. This group of features contains numerical values about other alerts in the same alerts stream that are semantically similar and semantically relevance to the alert in-question. We calculate these features using the raw alert attributes and the attack semantic. For each alert we analyze the surrounding alerts (alerts with detection timestamp less than, greater than, or equal to the current alert timestamp) looking for the semantically similar alerts.

We consider two alerts a_i and a_j to be related if they satisfy one of the following semantic relatedness conditions:

- C_1 : a_i and a_j share the same source IP address and destination IP address
- C_2 : a_i and a_j share the same source IP address and the same attack semantic
- C_3 : a_i and a_j share the same destination IP address and the same attack semantic

- C_4 : a_i and a_j share the same source IP address and destination IP address and their attacks have different impact and/or privileges.

For each alert a_i in a given alerts stream we process this alerts stream and count the number of alerts that satisfy at least one of the alert semantic relatedness conditions. The number of alerts that satisfy one condition is divided by the number of all similar alerts (for normalization purpose) and counts as one feature. We calculate the number of related alerts over different time windows. The time window is defined based on the average inter-arrival time between alerts.

5.2.2 Alert Verification Using Nearest Neighbors Algorithm

The first technique to verify IDS alerts and detect false positives uses the k-nearest neighbors (KNN) algorithm, and is very simple to implement. It begins with a set of labeled IDS alerts where each alert is labeled as either false positive or true positive, as illustrated in Table 5.4.

Index	Time	Src IP	Dst IP	Attack Signature	Label
1	1299078650	B.B.B.12	192.168.20.70	SID-19	FP
2	1299078650	B.B.B.12	192.168.20.70	SID-19	FP
3	1299078830	C.C.C.97	192.168.20.80	SID-27	TP
4	1299078950	C.C.C.21	192.168.20.80	SID-83	TP
5	1299079250	192.168.70.33	192.168.70.50	SID-08	FP
6	1299035930	192.168.70.25	192.168.70.50	SID-08	FP
7	1299036010	C.C.C.10	192.168.20.80	SID-54	FP
8	1299036138	C.C.C.10	192.168.20.80	SID-63	FP
9	1299288650	192.168.70.22	192.168.10.12	SID-31	FP
10	1299288770	192.168.70.25	192.168.10.12	SID-31	FP
11	1299289250	C.C.C.21	192.168.20.70	SID-83	TP

Table 5.4: Example of labeled raw IDS alerts

We calculate the alert context for each labeled alert in the training set. For instance, the alert context for the alert with index number 7 in Table 5.4 is shown in

Figure 5.6.

The diagram illustrates the structure of alert context features. A large bracket on the left groups the entire table as 'Alert Context'. A smaller bracket on the right groups the bottom four rows as 'Related Surrounding Alerts'. The table itself is organized into four semantic categories, each indicated by a bracket on the right side of the table:

- Raw Alert**: Time, Src IP, Dst IP, Attack
- Attack Semantic**: Impact, Vulnerability, Platform, Protocol, Service, Privileges
- Target Semantic**: Type, Countermeasure, Access, Platform, Service
- Related Surrounding Alerts**: C₁, C₂, C₃, C₄

Feature	Value
Time	1299036010
Src IP	C.C.C.10
Dst IP	192.168.20.80
Attack	SID-54
Impact	code-execution
Vulnerability	Implementation bug
Platform	Windows XP
Protocol	HTTP
Service	Apache 2.2
Privileges	any
Type	Web-Server
Countermeasure	Snort, IPTables
Access	any
Platform	Windows 7
Service	MS IIS v7.5
C ₁	0.25
C ₂	0.25
C ₃	0.5
C ₄	0.0

Figure 5.6: Example of alert context features

After calculating the context of each alert in the labeled alerts set, we use these labeled alerts to classify new unlabeled alerts. A new unlabeled alert can be classified as false positive or true positive based on its k nearest neighbors in the labeled alerts set. For any unlabeled raw alert we calculate the alert context and then calculate the semantic distance between the context of the unlabeled alert and each alert context in the labeled alert set. Note that in our approach we associate with each alert feature a concept tree in which the feature itself is the root node while the feature values correspond to the leaves of the tree.

The semantic distance between two alerts a_i and a_j is given by equation 5.2.1. Where $sim(a_i, a_j)$ is the semantic similarity between a_i and a_j , given in equation

4.5.3. Note that the more semantically similar two alerts are, the lower their semantic distance and vice-versa.

$$\text{SemanticDistance}(a_i, a_j) = 1 - \text{sim}(a_i, a_j) \quad (5.2.1)$$

To explain how the classification of new unlabeled alerts works, let us assume that the IDS in the target network example generated the three alerts in Table 5.5.

Index	Time	Src IP	Dst IP	Attack Signature	Label
1	1299078650	A.A.A.65	192.168.20.70	SID-95	??
2	1299078650	B.B.B.12	192.168.20.80	SID-54	??
3	1299078830	H.H.H.201	192.168.70.50	SID-27	??

Table 5.5: Example of unlabeled raw IDS alerts

Using the concept trees for the target network example in Section 5.1 we can calculate the semantic distance between each new unlabeled alert in Table 5.5 and each labeled alert in Table 5.4. For example the semantic distances between alert with index 1 in Table 5.5 and each labeled alert in Table 5.4 are shown in Table 5.6. The semantic distances in Table 5.6 are calculated based on five features (src IP, Dst IP, attack, target OS, running service) that represent the alert context.

Based on the semantic distance values in Table 5.6, and using nearest neighbors (where $k=1$ or $k=3$) the unlabeled alert will be classified as True Positive (TP). Note that alert 1 and alert 2 in Table 5.4 are duplicated alerts, therefore we consider them as one alert when we count the k -nearest neighbors.

Looking at our technique we can see that the proposed semantic distance metric assumes the alert context features have equal importance. However, this assumption is not always true as some of the alert context features may be more important than

the others. Moreover, from the intrusion analyst view, some of the context features are not important at all, specifically when they relate to some intrusion attempts. For this reason we suggest the use of a weighted semantic distance metric.

Labeled Alert	Semantic Distance	Predicated Label
alert 1	0.20	FP
alert 2	0.20	FP
alert 3	0.21	TP
alert 4	0.21	TP
alert 5	0.64	FP
alert 6	0.64	FP
alert 7	0.28	FP
alert 8	0.28	FP
alert 9	0.59	FP
alert 10	0.59	FP
alert 11	0.11	TP

Table 5.6: Semantic distances between unlabeled alert 1 in Table 5.5 and each labeled alert in Table 5.4

A weighted semantic distance metric will assign for each semantic feature a weight that represents the importance of the feature. Given an alert a_i let a_{ik} denote the k_{th} feature of alert a_i . We modify Equation 5.2.1 to include the weight of each semantic feature by defining the semantic distance of two alerts a_i and a_j as follows:

$$SemanticDistance(a_i, a_j) = 1 - \left(\frac{\sum_{k=1}^s w_k \cdot sim(a_{ik}, a_{jk})}{s} \right) \quad (5.2.2)$$

Where w_k is the weight assigned to feature k ($1 \leq k \leq s$).

5.2.3 Alert Verification Using Rule Induction

The k-nearest neighbor is known to be an expensive classifier. The runtime complexity for the KNN algorithm is $O(m \cdot n)$ where m is the number of features and n is the number of instances in the labeled training set. This is because we compute the distance between the unlabeled instance (alert) and every labeled instances (alerts) in the training set.

Considering the massive number of labeled/unlabeled alerts we need to deal with, a KNN classifier might not be the best choice for online alert verification. For this reason we propose a different technique to verify IDS alerts that eliminates false positives using rule induction [37].

Rule Induction

Rule induction is a machine learning method that learns classification rules from a set of labeled instances. It is another form of instance-based learning. The learning process with rule induction uses a set of training examples (labeled instances) to extract a set of rules that can classify new unlabeled instances. The classification rules are extracted by generalizing the training examples.

One of the main advantages of rule induction over other machine learning methods is the generated rules are easy to interpret and most of the time are human readable. The main components of a rule induction system are:

- Description Language: to describe the training dataset and the extracted rules.
- Training examples: a set of labeled instances that will be used to extract the classification rules.

- Cover function: a method to decide if an instance (labeled or unlabeled) is covered by the rule or not.

The extracted rule set should have two properties, namely, consistency and completeness. Consistency means each rule in the rule set should cover only one class (label) in the training set. Completeness means the rule set should cover all the examples in the training set. However, the above definition of consistency and completeness is very strict and unrealistic in any real learning problem. Most of the time if the rule set is complete, it is mostly not consistent and vice-versa. Therefore, usually the completeness and consistency properties are relaxed using a threshold or other form of measurements.

Ontology-based Rule Induction (ORBI)

Using ontologies and the theory of rule induction we propose a new rule induction algorithm to learn classification rules using an ontology and a set of training examples. The proposed algorithm is useful for learning classification rules where the domain of the learning problem is represented by an ontology. It is assumed that each feature in the training examples is associated with a concept tree (a taxonomy in the learning problem domain ontology). The root node of the concept tree is the feature itself while the feature values correspond to the leaves of the tree.

Using an ontology-based rule induction will allow us to transform the massive number of IDS alerts into a compressed set of rules. In our case, the description logic serves as the description language to express the rules for the rule induction system and the taxonomic relations serve as the cover function.

Our proposed ontology-based rule induction algorithm performs two basic operations to learn classification rules from training examples. The first operation is a generalization operation and the second operation is a fusion operation. The result

of the two operations is a new hybrid-alert that we can use as an alert classification rule. Usually each generated rule is evaluated based on its classification accuracy. If the classification accuracy is more than a predefined threshold the rule will be added to the rules set otherwise the rule will be dropped.

In general, a hybrid alert has the same format, and therefore the same types of attributes as a raw alert. The main difference between the attributes in a hybrid alert and those in the raw alert is the level of abstraction. Hybrid alert attribute values (i.e. concepts) will be equal or more abstract than corresponding raw alert attribute values.

Before we explain our ontology-based rule induction technique in details, let us illustrate the general idea of the technique. Let Table 5.7 represent the alert training set.

Src	Dst	SID	Service	OS	Label
C.C.C.97	192.168.20.70	SID-19	IIS-6.0	WinXP	FP
C.C.C.10	192.168.20.80	SID-83	IIS-7.5	Win7	FP
A.A.A.47	192.168.20.70	SID-27	IIS-6.0	WinXP	FP

Table 5.7: Example of alert training set for rule induction

Now, using the labeled alert set in Table 5.7 we can apply a simple generalization and fusion step to learn the classification rule. One possible generalization operation is to replace the values of each feature in the labeled alert set (excluding the label) by their lowest common ancestor (LCA)[2] in the concept tree associated with that feature. Then, we fuse redundant alerts (including their labels). By applying these two operations on the alerts in Table 5.7, it will result in one new alert (hybrid alert) that represents the alert set as illustrated in Table 6.4

Src	Dst	SID	Service	OS	Label
Egypt	Public-Web	Web-App-Exploit	MS IIS	WinNT	FP

Table 5.8: Alert training set after applying the OBRI technique

The generalization and fusion operations produce a high-level (hybrid) alert. We can think of this high-level alert as a classification rule. This classification rule says that for any given alert if the alert features satisfy a set of conditions then this alert is a false positive. The conditions are simply taxonomic relations instance-of and subclass-of. The conditions in this case are:

- **Egypt(?src)**: the source of the alert is an instance/subclass of the class Egypt.
- **Public-Web(?dst)**: the destination of the alert is an instance/subclass of the class Public-Web.
- **Web-App-Exploit(?sid)**: the attack reported in the alert is an instance/subclass of the class Web-App-Exploit.
- **MS-IIS(?service)**: the service attacked during the attack in the alert is an instance/subclass of the class MS-IIS.
- **WinNT(?os)**: the target operating system of the alert destination is an instance/subclass of the class WinNT.

Finally, this high-level alert can be rewritten as a classification rule R_1 using SWRL as follows:

$$\begin{aligned}
 &Alert(?x) \wedge Egypt(?c) \wedge PublicWeb(?v) \wedge WebAppExp(?a) \wedge WindowsNT(?o) \wedge \\
 &\quad IIS(?s) \wedge hasSource(?x, ?y) \wedge hasGeoLoc(?y, ?c) \wedge hasTarget(?x, ?z) \wedge \\
 &\quad hasVirLoc(?z, ?v) \wedge report(?x, ?a) \wedge hasOS(?z, ?o) \wedge hasService(?z, ?s) \rightarrow \\
 &\quad\quad\quad FalsePositive(?x)
 \end{aligned}$$

It is clear that the extracted classification rule is complete and consistent with respect to the training set in Table 5.7. In addition, the idea of ontology-based rule induction allows us to generalize beyond the examples that have been provided during the training phase. In other words the extracted classification rules can classify novel alerts that were not present during the training phase. For example, let us consider the alert in Table 5.9:

Src	Dst	SID	Service	OS	Label
C.C.C.21	192.168.20.80	SID-79	IIS-7.5	Win7	??

Table 5.9: Unlabeled novel alert example

The alert in Table 5.9 is a novel alert (i.e. not existing in the training set); however it is covered by the extracted classification rule. This means using the classification rule R_1 we can predict the class of this novel alert, which is false positive.

While we were able to learn a classification rule that is complete and consistent with respect to the training set, this is not always possible. Moreover, using the lowest common ancestor to generate the rule is not applicable with real alert set. In most cases using LCA will result in learning overgeneralized classification rules that are neither consistent nor complete and have poor accuracy. Therefore, it is important to design a new algorithm that can extract accurate classification rules using the OBRI approach.

A Brute-Force OBRI Algorithm

For a given training set of alerts A , we can learn the optimal (complete and consistent) set of classification rules R using a brute-force technique. In fact, the only method to find an optimal R for any given A is using a brute-force search. The runtime complexity of applying a brute-force search for finding R is $T(n) = n \cdot r$, where n

is the number of alerts in A , and r is the number of all possible classification rules. The number of all possible classification rules for a given alert set A is given by the following equation:

$$r = \prod_{i=1}^m |c_i| \quad (5.2.3)$$

Where c_i denote the set of classes involved in the concept tree associated with an alert feature of index i ; $|c_i|$ denote the cardinality of c_i and m denote the total number of alert features.

For instance, the number of all possible classification rules r for the alert set in Table 5.4 is $r = 14 \times 8 \times 13 \times 4 \times 13 = 75712$ rules. Using the concept trees in the Target network example and the training set we will have 14 concepts in the source feature, 8 concepts in the destination feature, 13 concepts in the attack feature, 4 concepts in the OS feature, and 13 concepts in the services feature.

Using a brute-force search we can find the optimal set of classification rules for the alerts in Table 5.4. By testing the 75712 possible classification rules we will end up with four rules that are consistent and complete as shown in Figure 5.7.

$$\begin{aligned}
 R_1 &= \text{Cairo}(\text{src}) \wedge \text{PubWeb}(\text{dst}) \wedge \text{WebAppExp}(\text{sid}) \wedge \text{WinNT}(\text{os}) \wedge \text{IIS}(\text{service}) \rightarrow \text{True Positive} \\
 R_2 &= \text{Cairo}(\text{src}) \wedge \text{PubWeb}(\text{dst}) \wedge \text{BOF}(\text{sid}) \wedge \text{WinNT}(\text{os}) \wedge \text{IIS}(\text{service}) \rightarrow \text{True Positive} \\
 R_3 &= \text{BC}(\text{src}) \wedge \text{PubWeb}(\text{dst}) \wedge \text{WebAppExp}(\text{sid}) \wedge \text{WinNT}(\text{os}) \wedge \text{IIS}(\text{service}) \rightarrow \text{False Positive} \\
 R_4 &= \text{Host}(\text{src}) \wedge \text{Private}(\text{dst}) \wedge \text{Attack}(\text{sid}) \wedge \text{WinNT}(\text{os}) \wedge \text{Service}(\text{service}) \rightarrow \text{False Positive}
 \end{aligned}$$

Figure 5.7: Alert Verification Rules

An Immune Inspired OBRI Algorithm

Using a brute-force method to learn the classification rules is computationally expensive. For this reason we have decided to design an OBRI algorithm using a computational intelligence technique inspired by the human immune system. Our algorithm can learn classification rules with an accepted level of completeness and consistency without the need to perform a brute-force search.

To learn classification rules and design an efficient ontology-based rule induction, we based our technique on the clonal selection theory. Clonal selection is the theory that explains how the human immune system responds to infection or antigens. In short when an antibody recognizes an antigen (nonself) with a certain affinity (degree of matching), this antibody is selected by the immune system proliferation. The objective of this proliferation process is to select those antibodies that can detect pathogens with a high detection rate. The immune system clones the selected antibodies and then the clones enter a mutation operation that results in new antibodies with higher affinity than the original antibodies. In other words, the generated antibodies after the proliferation are more capable of recognizing antigens than the original antibodies.

Before we explain our immune inspired technique let us review some basic immune terminologies and the general description of the clonal selection algorithm as explained by De Castro[14] .

- ***Affinity:*** a measurement to capture the degree of binding (relatedness) between an antibody (Ab) and antigen (Ag). The stronger the binding, the higher the affinity.
- ***Affinity Maturation:*** the process that increases the antibodies affinity for antigens through selection and hypermutation as part of the adaptive immune

response

- ***Antibody:*** a Y-shaped protein generated by the B-cells in response to antigen. Antibodies are used by the immune system to detect and eliminate antigens.
- ***Antigen:*** any micro-organism (pathogen or infectious agent) that when presented into the body triggers the immune system response (production of antibodies)
- ***Bone Marrow:*** a soft tissue, which as indicated by the name is located in the interior of bones. This soft tissue is the site responsible for the generation of all the blood cells including the immune cells (lymphocyte).
- ***Clonal Selection:*** the theory that explains how the immune system clones selected antibodies and mutates these antibodies to produce new antibodies with higher affinity.
- ***Clone:*** an organism or a group of cells that is genetically identical to another organism (ancestor) from which it was created.
- ***Gene:*** a hereditary unit of a living organism; consists of a sequence of DNA that defines a specific characteristic in the organism.
- ***Mutation:*** the process of changing the DNA sequence within a gene, which results in the creation of a new gene with new characteristics that do not exist in the original gene.

The general clonal selection algorithm can be described using the following basic steps:

1. Generate an initial set AB of antibodies.
2. Given set AG of antigens, for each antigen in AG.

- 2.1. Calculate the affinity between this antigen and each antibody in AB.
 - 2.2. Select a set X of antibodies, where X contains the antibodies with highest affinity in AB.
 - 2.3. For each antibody in X , clone this antibody m times. The number of clones depends on the affinity of the antibody; the higher the affinity the higher the number of clones and vice-versa.
 - 2.4. Mutate each generated clone in the previous step. The mutation rate also depends on the clone affinity; the higher the affinity the smaller the mutation rate and vice-versa.
 - 2.5. Add the mutated clones to AB; select the antibodies with highest affinity as the memory of the antigen.
 - 2.6. Select m number of antibodies with lower affinity in AB and replace them with X .
3. If stopping criteria are met, then stop, else go to step 2

Now, after explaining the clonal selection theory and the general algorithm, the key question is how do we use this computational technique to learn classification rules and eliminate false positive alerts. The first step to apply clonal selection is to represent our alert verification process as a clonal selection computational process. In other words, we need to map the alert verification steps and the immune system model. Table 5.10 summarizes the mapping between the alert verification process and the immune system.

As indicated in Table 5.10 each concept tree is a gene library. Since an alert message contain m features and each feature is associated with one concept tree then we have m gene libraries. We use these gene libraries to generate candidate classification rules (hybrid alert). We want to avoid the generation of overgeneralized

Alert Verification	Immune System
Hybrid Alert (Rule)	Antibody
Rule Coverage & Consistency	Affinity
False Alert	Antigen
True Alert	Self (body cell)
Concept Tree	Gene Library
Alerts Generalization	Bone Marrow, Affinity Maturation, and Mutation

Table 5.10: Mapping Between Alert Verification and Immune System

classification rules. To achieve that, we do not use the entire concept tree in the ontology as a gene library but only a subtree of this concept tree is used as the gene library for a given alert feature. This subtree is extracted from the original concept tree in the ontology by applying two simple rules as follows.

R1: The root of the subtree for a given feature f is the highest common ancestor of all the values of f in the training set.

R2: Any concept in the subtree must be an ancestor for at least one feature value in the training set.

Using the labeled alert set in Table 5.4 and the concept tree in Figure 5.4 we can extract the subtree for any feature in the alert message. For example, let us consider the OS feature; the subtree for the OS feature includes the following concepts: *WinNT*, *Win7*, *WinXP*, and *Win2000*. Limiting the gene library for the OS feature to these four concepts means any generated rule that say anything about the OS will only contain one of these four concepts. This prevents our rule induction algorithm from generating overgeneralized rules that are not supported by any training example. In addition, it reduces the size of the gene library and therefore the rule space which will improve the runtime of the algorithm.

In our OBRI technique we use the gene library to generate candidate antibodies. Each candidate antibody is tested against the training set. Those candidate antibod-

ies that show a specific level of affinity with the training set are selected as mature antibodies. The set of mature antibodies represents the learned classification rules. Generating the set of mature antibodies or the classification rules requires that we define a method for calculating the affinity and another method to perform affinity maturation.

The goal of the rule induction is to learn a set of rules that is consistent (high accuracy) and complete (high coverage). For this reason the affinity of the antibody is determined based on the accuracy and the coverage of the antibody. The accuracy and coverage of an antibody ab for a given class X are given by the following equations:

$$accuracy(ab, X) = \frac{cardinality(ab \rightarrow X, E)}{cardinality(ab, E)} \leq 1 \quad (5.2.4)$$

$$coverage(ab, X) = \frac{cardinality(ab \rightarrow X, E)}{cardinality(X, E)} \leq 1 \quad (5.2.5)$$

Where $cardinality(C, S)$ is the number of examples in set S that satisfy condition C . For example, $cardinality(ab \rightarrow X, E)$ is the number of the examples in the training set E that are covered by ab and belong to the class X . Given the accuracy and coverage of an antibody ab , we can calculate the affinity of ab with X by the following equation:

$$affinity(ab, X) = w_a \cdot accuracy(ab, X) + w_c \cdot coverage(ab, X) \quad (5.2.6)$$

Where w_a and w_c are weights assigned to accuracy and coverage, respectively and $w_a + w_c = 1$

When we calculate the affinity between ab and X , we use weights w_a and w_c to control how the accuracy and the coverage affect the overall affinity. The affinity is a

value between 0 and 1, where 0 indicates no affinity at all and 1 indicates maximum affinity.

To perform affinity maturation we need to select a subset of the candidate antibodies and mutate these antibodies to obtain mature antibodies (i.e, have higher affinity with X). It is possible to select n number of antibodies with highest affinity or to select all the antibodies with an affinity greater than a predefined maturation threshold. Then, the selected antibodies are cloned and mutated. Each selected candidate antibody is cloned for m times based on the antibody affinity, the length of the antibody, and the size of the gene libraries. The number of clones that will be produced for each antibody ab is calculated using equation 5.2.7.

$$clone(ab_i) = \left\lfloor \frac{L \times |G|}{i} \right\rfloor, (1 \leq i \leq n) \quad (5.2.7)$$

Where ab_i is the i^{th} antibody in n number of selected antibodies, and $1 \leq i \leq n$. These selected antibodies are sorted based on their affinity, such that ab_1 is the antibody with the maximum affinity and ab_n is the antibody with the minimum affinity. Finally, L is the length of the antibody (the number of alert features) and G is the sum of sizes of the gene libraries (the total number of concepts in the concept trees). Note that during the generation of the antibodies we update the gene libraries by removing genes that are completely covered. In other words we continue pruning the concept trees and remove the concepts that are completely covered by the matured antibodies. This means we always make sure that any concept in the concept trees (gene libraries) is at least an ancestor for one feature value in the uncovered training examples.

After cloning the selected antibodies; the next step in the affinity maturation is to mutate the clone in order to produce matured antibodies with higher affinity. The mutation is usually a random process where we randomly select some genes (alert fea-

tures) and replace them with other genes from the gene libraries (concepts from the concept trees). The strength of the mutation depends, in general, on the affinity and can be guided using the coverage and the accuracy of the cloned antibody. For example, if the coverage of the antibody is low and the accuracy is high then the mutation tends to generalize the genes in the antibody (superclasses replace subclasses) and if the coverage is high and the accuracy is low then the mutation tends to specialize the genes in the antibody (subclasses replace superclasses).

When the affinity maturation completes, we calculate the affinity of the mutated antibodies and add them to the original antibodies population. Then, we select the n antibodies with the maximum affinity or greater than a predefined affinity threshold. Any training example in the training set that is covered by the selected antibodies will be removed from the training set. We continue repeating the clonal selection process as we described until all the training examples are covered or in other word until the training set is empty.

At the end of the clonal selection we will have a set of matured antibodies (classification rules) that cover all the examples (alerts) in the training set. At this point, it is possible that some of the generated rules are semantically redundant. Two classification rules r_i and r_j are semantically redundant if r_i is covered by r_j or r_j is covered by r_i . This means the training examples covered by one rule are covered by the second rule. The way to deal with semantically redundant classification rules is by looking at the rule accuracy and coverage. We always select the most generalized rule if it has better or the same accuracy as the least generalized rule. A less generalized rule will be selected over a more generalized one if it has better accuracy and dropping the more generalized rule does not prevent satisfying the required level of coverage. The main steps of our semantic similarity based alerts verification process are illustrated by Algorithm 1.

Algorithm 1: Ontology-based Rule Induction

```

/*  $E$  a set of labeled alerts (training examples) */
/*  $R$  a set of alert classification rules */
Input:  $E$ 
Output:  $R$ 
1 begin
2    $R \leftarrow \emptyset$ ;
3   while  $E$  has more Examples do
4      $P \leftarrow$  generate antibodies population;
5     affinity( $P, E$ );
6     Let be  $S$  a subset of  $P$  with maximum affinity;
7     foreach  $ab$  in  $S$  do
8        $C \leftarrow$  clone( $ab$ )
9     end
10    foreach  $clone$  in  $C$  do
11       $X \leftarrow$  Mutate ( $clone$ )
12    end
13    affinity( $X, E$ );
14    insert( $X, P$ );
15    Let  $S$  a subset of  $P$  with maximum affinity;
16    foreach  $e$  in  $E$  do
17      if  $e$  covered by  $S$  then
18        remove  $e$  from  $E$ ;
19      end
20    end
21    insert( $S, R$ )
22  end
23   $R \leftarrow$  refine( $R$ );
24  return  $R$ ;
25 end

```

The input to our ontology-based rule induction algorithm is E , a set of labeled alerts (training examples). The output of the algorithm is R , a set of alert classification rules. The algorithm terminates when it generates rules that cover all the examples in the training set. In every iteration the algorithm generates P , a population of candidate antibodies. Each antibody in P is evaluated with E to calculate its affinity. Then, a set S of antibodies in P with maximum affinity are selected for affinity maturation. Each antibody selected for affinity maturation is cloned and its clones are added to the set C and then each cloned antibody is mutated and added to the set X . After that, the algorithm calculates the affinity of each mutated clone and adds all the clones in X to the original population P . Again, the algorithm selects the antibodies with the highest affinity and puts them in the set S . Next, each alert (example) in the training set E that is covered by at least one antibody from S is removed from E . At the end of the iteration, the antibodies in S are added to the

set R . Finally, when all the examples are covered, we refine the set R to eliminate semantically redundant rules and return the refined set of rules R .

5.3 IDS Alert Aggregation

We think of the problem of alert aggregation or alert fusion as a special case of automatic text summarization problem [18]. In automatic text summarization we usually have one or more sources of information and we try to produce a summary that retains the most important information. In general, there are two main methods for automatic summarization, namely, extraction and abstraction. Extraction methods identify important knowledge or key phrases in the text and use them as a summary. Abstraction methods on the other hand produce important information and key points in a new way. Usually the quality of summary generated by abstraction methods is better than the one generated by extraction methods. However, developing abstraction methods for automatic summarization is harder than extraction methods and usually requires the use of artificial intelligence and natural language processing techniques.

We propose an automatic alert aggregation and summarization technique. Our technique is an abstraction-based technique. We process the raw IDS alerts (regardless of their true/false labels) and generate hybrid alerts. Each generated hybrid alert fuse/summarize a set of semantically similar alerts. Our technique relies on the intrusion ontology to generate the hybrid alerts. In particular, raw alerts that share semantic similarity are fused together into a hybrid alert that summarizes these raw alerts.

The basic operations of the alert aggregation are similar to the basic operations of the alert verification, namely, generalization and fusion, that produce a hybrid alert.

This hybrid alert is a high level alert that summarizes a set of raw alerts. To illustrate our alert aggregation technique, let assume we have the set of raw IDS alert in Table 5.11.

Index	Src	Dst	SID	Service	OS
1	A.A.A.97	192.168.20.70	SID-27	IIS-6.0	WinXP
2	A.A.A.65	192.168.20.80	SID-19	IIS-7.5	Win7
3	A.A.A.65	192.168.20.70	SID-83	IIS-6.0	WinXP
4	A.A.A.97	192.168.20.80	SID-19	IIS-7.5	Win7
5	A.A.A.65	192.168.20.80	SID-79	IIS-7.5	Win7

Table 5.11: Raw IDS Alerts before Aggregation

Aggregating the raw alerts in Table 5.11 can be achieved by generalizing and fusing the alerts. For instance, we can aggregate this subset of alerts into one hybrid alert, as in Table 5.12 or into 2 hybrid alerts as in Table 5.13. As we can see from Table 5.12 and Table 5.13 there are at least two different possible aggregations for the raw alerts in Table 5.11.

Src	Dst	SID	Service	OS
Alex	Public-Web	Web-App-Exploit	MS IIS	WinNT

Table 5.12: Summarizing Raw Alerts Using One Hybrid Alert

Src	Dst	SID	Service	OS
Alex	192.168.20.70	Dir-List	IIS 6.0	Win XP
Alex	192.168.20.80	XSS	IIS 7.5	Win 7

Table 5.13: Summarizing Raw Alerts Using Two Hybrid Alerts

Now, how can we select the appropriate aggregation results for the raw alerts in Table 5.11 ? In other words, how can we measure the quality of the generated hybrid alerts ? In general, the alert aggregation is usually evaluated using the *alert reduction rate (ARR)*, which is computed as the difference between the original number of alerts and the alerts remaining at the end of the aggregation process over the original number of alerts. The alert reduction rate (ARR) is given by the following equation.

$$ARR = 1 - \frac{|H|}{|A|} \quad (5.3.1)$$

Where H is the set of hybrid alerts resulting from the aggregation process and A is the original set of raw alerts before the aggregation. Despite its popularity, we believe, however, that the ARR is not enough to evaluate the effectiveness of the aggregation process. In fact the ARR captures well the quantitative aspect of the alert aggregation process but misses altogether the qualitative perspective. The use of one hybrid alert to aggregate the raw alerts in Table 5.11 yields better ARR than using two hybrid alerts. However, it is clear that the use of two hybrid alerts as in Table 5.13 is more informative. For example, when using two hybrid alerts we know that only the web server with the IP address 192.168.20.80, was attacked by XSS attacks. To bridge this gap, we assess the quality of our aggregation process by measuring objectively the *information loss* occurring during this process.

The main challenge with alert aggregation is how we control the generalization and fusion operations. In the alert verification, this was less complex because the alerts were labeled either false positives or true positives. In other words, the alerts were sorted into two classes. Therefore, in the alert verification we try to generate hybrid alerts that do not fuse alerts from different classes. This means the labels of the alerts play a major role in controlling the generalization and the fusion of the

alerts. Moreover, the end objective of the alert verification is to learn a set of rules that we can use in the future to classify alerts. On the other hand, in the alert aggregation there is no learning; the generated hybrid alerts summarize a specific set of raw alerts and is not suitable to describe another set of raw alerts. In addition, not all the alert context features are necessary for aggregating the alert. For example, the features from the related surrounding alerts (alert context feature) are not important. Therefore, we believe that for aggregation purpose it is enough to use only the raw alert features in addition to some selected features from the attack semantic and target semantic.

5.3.1 A Lightweight Alert Aggregation Method

If our objective is to aggregate raw alerts to achieve a specific alert reduction rate like previous work in the area, then using the intrusion ontology we can develop an efficient lightweight alert aggregation algorithm. The proposed algorithm applies a hill climbing technique to generalize and fuse the raw alerts. This lightweight aggregation algorithm only guarantees that the raw alerts will be aggregated to reach a specific alert reduction rate. It does not consider the similarity between the alert or the information loss rate when it performs the aggregation.

To explain the main idea of our lightweight alert aggregation algorithm let us say that we have n number of raw IDS alerts. Each alert consists of m number of features. Again, each feature is associated with a concept tree where the feature itself is the root node while the feature values correspond to the leaves of the tree. Now, let us assume that we want to aggregate the raw alerts and reduce them to u number of hybrid alerts, where $u < n$. To reach the target alert reduction u we process the raw alerts as follows.

Because we want to reach a specific alert reduction rate by reducing the number

of alerts from n to u , we must make sure that each alert feature has only u number of unique feature values. This is because if the number of unique feature values for any alert feature is greater than u , then the total number of alerts must be greater than u . To reduce the number of unique feature values in each feature to u or less, our technique iterates over each alert feature and performs the two basic operations of generalization and fusion. In the generalization operation, each feature value is replaced by its least ancestor in the concept tree associated with that feature. After each generalization operation, we check the alert set and fuse redundant alerts.

After reducing the number of unique feature values in each alert feature to u , it is possible that the total number of remaining alerts is more than u . This means we did not reach the required reduction rate. This also means further generalization and fusion operations are required. Because each alert feature has only u unique feature values, then it is possible to select one or more alert features to perform additional generalization until we reach the desired alert reduction. It is important that we do not perform all the additional generalization operations on the same alert feature, as, this mostly will result in overgeneralizing this alert feature. Therefore, in each iteration we should select the alert feature that has the larger number of unique feature values or the one whose concept tree has the maximum depth. The main steps of our lightweight alert aggregation are summarized in Algorithm 2

As mentioned before, the lightweight aggregation algorithm focuses only on achieving a specific alert reduction rate. In general, it does not give any promise on the quality of the aggregation process. To avoid overgeneralization, the lightweight aggregation algorithm always starts by the feature value with the maximum depth in the concept tree. In addition, when the number of unique feature values is less than the desired alert reduction value it will select the feature with the maximum number of unique values whose concept tree has the maximum depth.

Algorithm 2: Lightweight Alert Aggregation

```

/*  $A$  a set of raw IDS alerts (training examples) */
/*  $H$  a set of hybrid IDS alerts (aggregated alerts) */
/*  $r$  the desired alert reduction */
Input:  $A$ 
Output:  $H$ 
1 begin
2   foreach feature  $f_i$  in  $F$  do
3     while number of unique feature values in  $f_i > r$  do
4       let  $v$  the feature value with maximum depth ;
5       replace  $v$  by its least ancestor in concept tree;
6       fuse redundant alerts in  $A$  ;
7     end
8   end
9   while number of alerts in  $A > r$  do
10    select feature  $f$  where  $f$  is a good candidate for generalization ;
11    let  $v$  the feature value with maximum depth ;
12    replace  $v$  by its least ancestor in concept tree;
13    fuse redundant alerts in  $A$  ;
14  end
15   $H \leftarrow A$ ;
16  return  $H$ ;
17 end

```

5.3.2 Alerts Aggregation Using Semantic Similarity

Despite the fact that lightweight alert aggregation algorithm can summarize raw IDS alerts and reach the target alert reduction rate, it does not really take full advantage of the intrusion ontology. One way to improve the alert aggregation process is to aggregate alerts based on their semantic similarity. This means a group of raw IDS alert will be fused into one hybrid alert if only their semantic similarity is no less than a given threshold.

Using the intrusion ontology and the semantic similarity metrics introduced in Chapter 4, we can measure the semantic similarity between a group of alerts. Then, if the semantic similarity between these alerts is greater than a given threshold, we can fuse these alerts into one hybrid alert. In other words, each hybrid alert will summarize a subset of the raw IDS alerts that share the same meaning (semantic). To aggregate the raw alerts using their semantic similarity we need to cluster the alerts first using their semantic similarity.

The main steps of our semantic similarity based alerts aggregation process are

illustrated by Algorithm 3. The algorithm performs two main operations, namely, clustering and fusion. The clustering operation groups semantically similar alerts into a single cluster based on a predefined similarity threshold. The fusion operation fuses the alerts that belong to the same cluster and generates a corresponding hybrid alert.

Algorithm 3: IDS Alerts Aggregation Algorithm	
1	begin
	/* A a set of intrusion alerts of size n */
	/* T semantic similarity threshold vector of size p */
	Input: A, T
	Output: H
2	/* Th : threshold vector of size p */
3	/* C : set of alerts clusters */
4	$Th \leftarrow [1, \dots, 1]$;
5	$i \leftarrow 0$;
6	$A' \leftarrow A$;
7	$C \leftarrow \emptyset$;
8	while $i \leq p$ do
9	for $j = 1$ to n do
10	$x \leftarrow true$;
11	if $C \neq \emptyset$ then
12	for $s = 1$ to $size(C)$ do
13	$c \leftarrow C[s]$;
14	$h \leftarrow$ hybrid-alert of cluster c ;
15	/* $h = [h_1, \dots, h_p]$, where h_s is the s^{th} attribute of h */
16	for $l = 1$ to p do
17	if $(sim(a_{j_l}, h_l)) \leq Th[l]$ then
18	$x \leftarrow false$;
19	break ;
20	end
21	end
22	if $x = true$ then
23	$c \leftarrow c \cup \{a_j\}$;
24	$h \leftarrow$ fuse a_j with h ;
25	$H \leftarrow H \cup \{h\}$;
26	break ;
27	end
28	end
29	if $(x = false)$ or $(C = \emptyset)$ then
30	let c new cluster such that $c = \{a_j\}$;
31	let h hybrid-alert of c such that $h = a_j$;
32	$H \leftarrow H \cup \{h\}$;
33	end
34	end
35	$A' \leftarrow H$;
36	$i \leftarrow i + 1$;
37	$Th[i] \leftarrow T[i]$;
38	end
39	return H ;
40	end

The algorithm takes two inputs. The first input is a set of raw IDS alerts sorted

by increasing order of occurrence time. The second input is a thresholds vector, where each element represents a predefined semantic similarity threshold for one of the symbolic attributes.

The output of the algorithm is a set of hybrid alerts that represents the original set of raw IDS alerts. In our work, an hybrid alert has the same format, and therefore the same types of attributes as a raw alert. The main difference between the attributes in a hybrid alert and those in the raw alert is the level of abstraction. Hybrid alerts' attributes values (i.e. concepts) will be equal or more abstract than corresponding raw alerts' attributes values. In addition, we associate with each hybrid alert its own information loss rate which depends on the level of abstraction of its attributes values.

The algorithm performs several rounds; during each round the alerts are grouped into one or more clusters and one hybrid alert is generated for each cluster. An alert will be assigned to a cluster if the attributes similarities between the alert and the hybrid-alert that represent this cluster are greater than some predefined thresholds.

Each time an alert is assigned to a cluster, we fuse that alert with the hybrid alert that represents this cluster and regenerate the hybrid alert of the cluster. The hybrid alert is regenerated by fusing the attributes of the hybrid alert with the attributes of the new alert. The fusion of two attributes from two different alerts consists of replacing them with their least common ancestor in their concept tree in the ontology. For example, let us assume that we have two alerts a_1 and a_2 . If we fuse the attribute attack-type where the value of that attribute in a_1 is *IISDir - List* and in a_2 is *ApacheDir - List* then the result will be *HTTPDirectory - List*, which is their least common ancestor according to the concept tree in Figure 4.8. At the end of each round, the hybrid alerts along with the remaining alerts (not aggregated yet) are sorted and passed as input to the next round of the algorithm and go through the same process outlined above.

The different rounds of the algorithm are determined by the similarity threshold vector used in the clustering. The rounds are designed so as to aggregate first the alerts that are most likely to have greater semantic similarity, and by setting the similarity threshold vector accordingly. This is performed by clustering the alerts for which a subset of attributes match perfectly (i.e. threshold = 1). The clustering is carried out iteratively by decreasing in each iteration the required number of alerts attributes that match perfectly, and lowering the thresholds for the remaining attributes to predefined levels. Hence, while in the first round the similarity thresholds are all set to one, in the last round they are set to the predefined values provided as input to the algorithm.

5.3.3 Information Loss Metric

When we summarize IDS alerts or any security log data, it is important to avoid losing security relevant data. Alert aggregation is like any other data aggregation process at least when it comes to information loss. Any process that gathers and expresses data in a summary form, results in information loss. Therefore, it is important to measure the amount of information loss resulting from aggregating the raw alerts into hybrid alerts. Measuring the amount of information loss in aggregated data is a complex problem. However, because of the existence of an ontological representation for the network intrusion domain, measuring information loss becomes much easier

In our aggregation technique, two concepts that belong to the same concept tree are aggregated by replacing them by their least common ancestor (LCA) from the corresponding concept tree. This will lead unavoidably to loss of information. To capture the information loss we need to measure the amount of information represented by each concept or class in the ontology. The difference between the amount of information of concept c_1 and its subclass c_2 represents the information loss occurred

by replacing c_2 by c_1 . The information content (IC) of a concept c can be used to measure the amount of information represented by c . Recently several approaches inspired by information theory have been proposed to measure the IC of a given concept in an ontology based on the taxonomic structure of the concept within the ontology [72, 73]. We use in our work the IC metric proposed by Sánchez and colleagues [72] and defined as follows:

$$IC(c) = -\log \left(\frac{\frac{|leaves(c)|}{|subsumers(c)|} + 1}{maxleaves + 1} \right) \quad (5.3.2)$$

Where $subsumers(c)$ is a function that returns the set of subsumers concepts of concept c (these include concept c as well as all its parents concepts), $leaves(c)$ is a function that returns all the leaf concepts that are subclasses of concept c , and $maxleaves$ is the total number of leaf concepts of the concept tree. The subsumers of a given concept and the leaves of that concept reflect the information content of that concept. Based on the principle of cognitive saliency¹, concepts are specialized when it is necessary to differentiate them from already existing ones [72]. So, concepts with more sub-concepts provide less information than concepts at lower levels of the hierarchy (such as leaf concepts).

Now, given a set of concepts C , we define the information loss rate (ILR) resulting from replacing the concepts in C by their least common ancestor a as follows:

$$ILR(C) = \frac{\sum_{c \in C} (IC(c) - IC(LCA(C)))}{\sum_{c \in C} IC(c)} \quad (5.3.3)$$

¹The salience or saliency of an object or a concept corresponds to its relative standing or quality with respect to its neighbors.

Where $LCA(C)$ corresponds to the least common ancestor of the concepts in C .

Using the above formula and given two alerts $a_i = [a_{i1}, \dots, a_{ip}]$ and $a_j = [a_{j1}, \dots, a_{jp}]$, the information loss rate occurring from aggregating symbolic attribute values a_{ik} and a_{jk} ($1 \leq k \leq s$) is defined as shown in equation ???. The information loss rate is a value between $\mathbf{0}$ and $\mathbf{1}$, where 0 corresponds to no information lost and 1 corresponds to 100% loss of information.

The information loss rate resulting from the aggregation of a set of alerts into a hybrid alert h is computed as the summation of the information loss rate of each attribute of H over the total number of attributes in h .

The information loss rate for an entire aggregation process generating a set of hybrid alerts H may be obtained as the average of the information loss rate over all the hybrid alerts in H . However, we take in this work a more conservative approach by defining the information loss rate for an entire aggregation process as the maximum information loss over the hybrid alerts involved in H .

Now, let us assume that we want to calculate the information loss rate resulting from aggregating the two classes $BOF - Web$ and $Web - App - Exp$ from the attack concept tree in Figure 5.2. The first common ancestor of the two classes in the ontology is the class $Web - Exp$. Using equation 5.3.2, we obtain the following: $IC(BOF-Web)=1.3$, $IC(Web-App-Exp)=1.2$ $IC(Web-Exp)=0.96$. Using the above metric, the information loss rate from aggregating the two classes $BOF - Web$ and $Web - App - Exp$ is 0.23.

5.4 Attack Scenario Reconstruction

The alerts resulting from the previous phases (alert verification and alert aggregation) are grouped into several clusters based on their semantic relevance. The obtained

clusters are analyzed using semantic inference to detect the causality relation between corresponding alerts. Then, the attack scenarios are extracted using semantic inference.

5.4.1 Semantic-based Alerts Clustering

The objective of semantic-based alerts clustering is to find groups of alerts that are semantically relevant with respect to particular attack scenarios. A cluster of semantically relevant alerts represents a candidate attack scenario. Given a set A of n number of alerts there are $2^n - 1$ possible alerts groupings, where each alert grouping corresponds to a candidate attack scenario. A generated candidate attack scenario may correspond to a true or false attack scenario.

Based on the inferred relations between alerts, we calculate the semantic relevance between them and construct what we refer to in this work as the alerts correlation graph (ACG). The ACG is an undirected weighted graph $G = (V, E)$, where V is a set of vertices representing alerts and E is a set of edges representing the relations between alerts. The edges in the ACG are labeled by the values of the semantic relevance between the alerts corresponding to adjacent vertices.

We illustrate the ACG concept through an example based on the set of alerts given in Table 5.14.

We will assume, in this example, that only three types of relations can occur between any two alerts, namely, *hasSameSource*, *hasSameTarget* and *report-SameAttack*, and also that each relation has a weight value equal 1.

We also restrict for simplicity sake the semantic of the inferred relations to equality. For instance, *hasSameSource* relation between a pair of alerts means their source IP addresses have equal values. But, we can redefine the semantic of the *has-*

ID	Source	Target	Attack
a_1	B.B.B.12	192.168.20.70	SID-03
a_2	B.B.B.12	192.168.20.80	SID-03
a_3	H.H.H.78	192.168.20.80	SID-13
a_4	B.B.B.12	192.168.20.70	SID-54
a_5	H.H.H.78	192.168.20.80	SID-03
a_6	B.B.B.12	192.168.20.70	SID-63
a_7	H.H.H.78	192.168.20.80	SID-19

Table 5.14: Alerts Examples

SameSource relation by considering two IP addresses to be relevant if they belong to the same subnet, domain, or even geographical location. Similarly, the semantic of the *reportSameAttack* relation can be extended from a strict definition where reported attacks (by related alerts) are exactly the same to a broader definition where the reported attacks have similar impact (e.g. loss of availability), affect similar assets (e.g. Web Server), or require the same privilege.

The semantics of the inferred relations affect the structure of the alerts correlation graph and can be used to find similarity between attack patterns or scenarios.

In the example, the maximum number of relations between any two alerts is 3, since as indicated above there are exactly three possible types of relations between the alerts.

Based on the above considerations, the constructed ACG for the alerts set in Table 5.14 is shown in Figure 5.8.

The edges of the ACG in Figure 5.8 are labelled by the semantic relevance values between corresponding alerts. For instance, alerts a_1 and a_6 being linked by two relations (i.e. *hasSameSource* and *hasSameTarget*), the semantic relevance between them is 2/3.

Algorithm 4 illustrates the steps to build the Alerts Correlation Graph. The algorithm takes a set A of hybrid or commonly formatted alerts as an input and

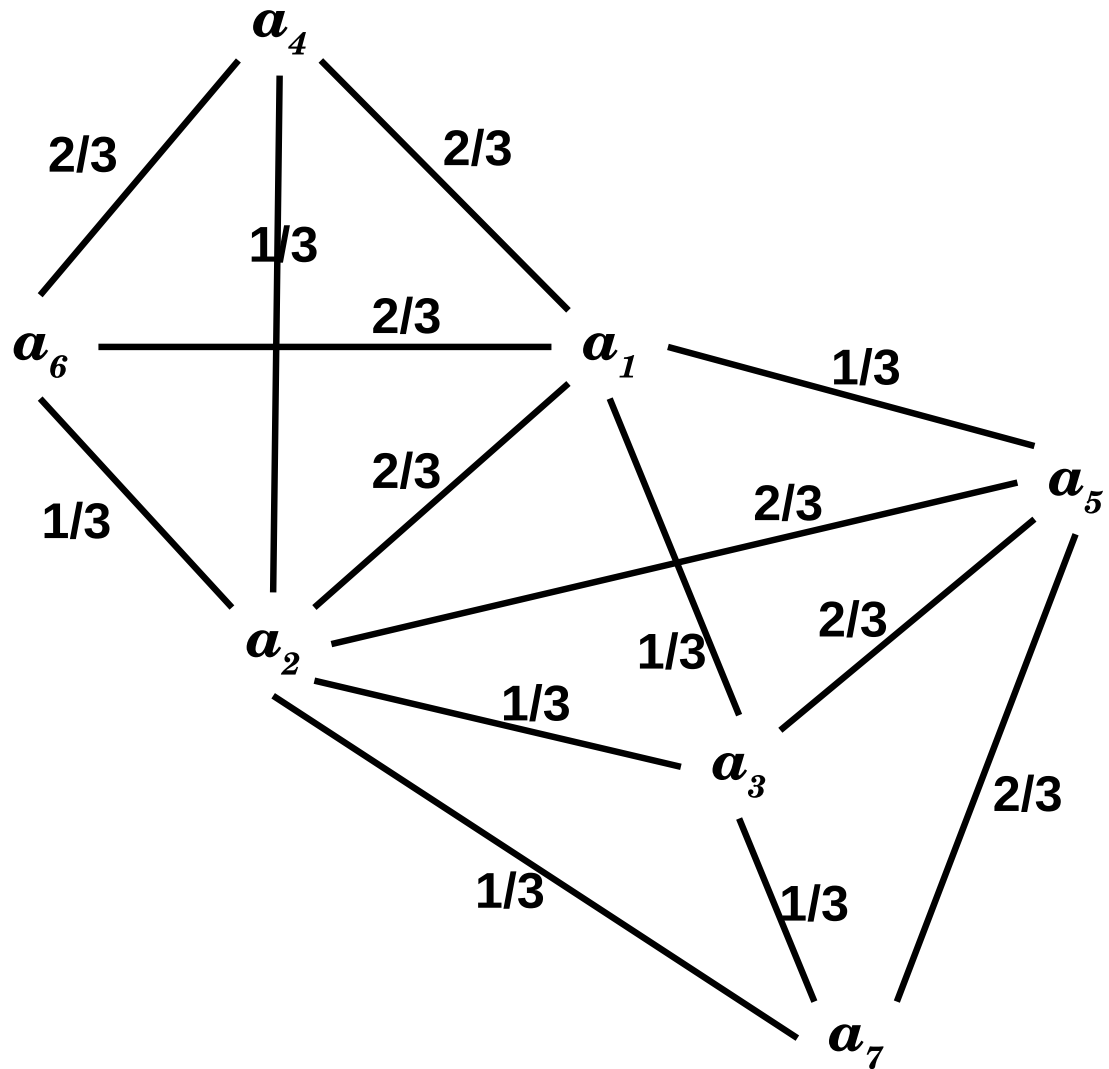


Figure 5.8: Example of Alerts Correlation Graph

generates the alerts correlation graph as an $n \times n$ matrix G where n is the total number of alerts in A . The entry $G[i, j]$ is zero if the semantic relevance between alerts a_i and a_j in A is less than a predefined semantic relevance threshold θ . If the semantic relevance value w is greater than or equal θ the algorithm set the value of $G[i, j]$ equal to w , which indicates that there is an edge e between a_i and a_j in G with weight w . The runtime complexity of Algorithm 4 is $O(n^2)$.

Algorithm 4: Constructing Alerts Correlation Graph

```

/* A a set of IDS alerts                                     */
/* G a matrix representing the ACG                         */
/* w semantic relevance measure between a pair of alerts in A */
/* θ semantic relevance threshold                         */
/* n number of alerts in A                               */
Input: A, θ
Output: G
1 begin
2   for i ← 1 to n - 1 do
3     for j ← i + 1 to n do
4       w ← sem_rel(ai, aj);
5       if w ≥ θ then
6         G[i, j] ← w;
7       end
8     end
9   end
10  return G;
11 end

```

The constructed ACG is used to extract candidate attack scenarios. In graph theory a clique in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge. In our case a clique in the ACG represents a subset of semantically relevant alerts. Therefore, we consider every maximum clique in the ACG as a candidate attack scenario or attack pattern. We use the well-known Bron-Kerbosch algorithm to find all maximum cliques in the ACG. In the ACG shown in Figure 5.8, there are three maximum cliques as illustrated by Figure 5.9.

Now, let c_1 , c_2 and c_3 denote the three maximum cliques in the ACG of Figure 5.9, where $c_1 = \{a_1, a_2, a_4, a_6\}$, $c_2 = \{a_1, a_2, a_3, a_5\}$ and $c_3 = \{a_2, a_3, a_5, a_7\}$. By looking closely at the above three candidate attack scenarios, we notice that they have some

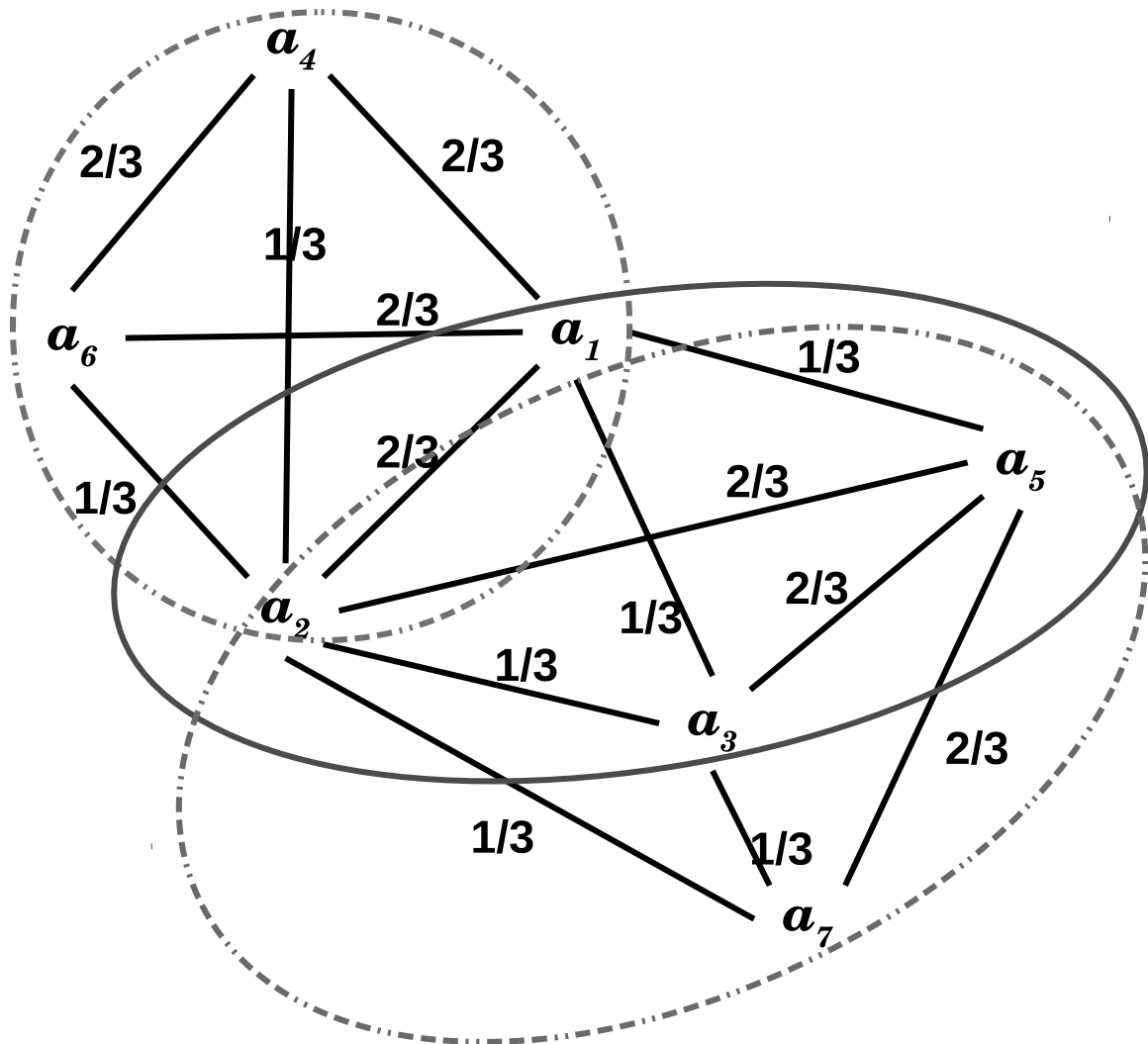


Figure 5.9: Maximum Cliques in an Alerts Correlation Graph

common vertices (alerts). For example, a_2 belongs to all three of them. Considering that an alert can belong to only one attack scenario, we need to refine our set of candidate attack scenarios by removing common alerts between them.

To remove a common alert from different candidate attack scenarios, we calculate the total semantic relevance of the common alert with respect to each candidate attack scenario, and assign it to the candidate attack scenario yielding the maximum total semantic relevance. This process will be repeated until each alert is assigned to only one candidate attack scenario.

The total semantic relevance of an alert with respect to a specific attack scenario is the sum of the semantic relevance between this alert and other alerts in the same attack scenario. For example, in Figure 5.9 the total semantic relevances of vertex a_1 in c_1 and c_2 are $(2/3 + 2/3 + 2/3 = 2)$ and $(2/3 + 1/3 + 1/3 = 1.3)$, respectively. Therefore, a_1 will be removed from c_2 and reassigned to only c_1 . By applying the same method to other common vertices, we will end up with only two candidate attack scenarios s_1 and s_2 , where $s_1 = \{a_1, a_2, a_4, a_6\}$ and $s_2 = \{a_3, a_5, a_7\}$.

Algorithm 5 illustrates the main steps to extract the candidate attack scenarios from an alert correlation graph. The algorithm takes as input an alert correlation graph G generated by Algorithm 4. First, the set C of maximum cliques are extracted from G using the Bron-Kerbosch algorithm. The alerts (or vertices) in each clique are sorted based on the alert number. To detect alerts that belong to more than one clique we apply a simple set intersection method, where each clique in C is treated as a set. The set intersection returns a list A' of alerts (vertices) that belong to more than one clique. Then, the algorithm iterates for n times, where n is the total number of alerts in A' . In each iteration the algorithm calculates the alert membership to each clique in C based on the total semantic relevance. At the end of each iteration an alert a is assigned to a clique c , where the membership of a with c is maximum.

scenarios, and l is the number of candidate attack scenarios in ACG.

5.4.2 Attack Causality Analysis

The main outcome of the attack scenario reconstruction process is the identification of the sequence of steps and actions taken by the intruder to breach the system. Our candidate attack scenarios generated from the semantic clustering phase lack such information. An effective technique to elicit the attack sequence consists of analyzing the causality between the individual attacks reported in the IDS alerts.

Existing attack scenario construction approaches use attack prerequisites and consequences to establish the causality between attacks or alerts. The attack prerequisites are the set of logical conditions to be satisfied for the attack to succeed while the attack consequences are the set of logical conditions that will become true when the attack succeeds. Two attacks a and b are causally related if at least one of the consequences of one of them is among the prerequisites of the second one.

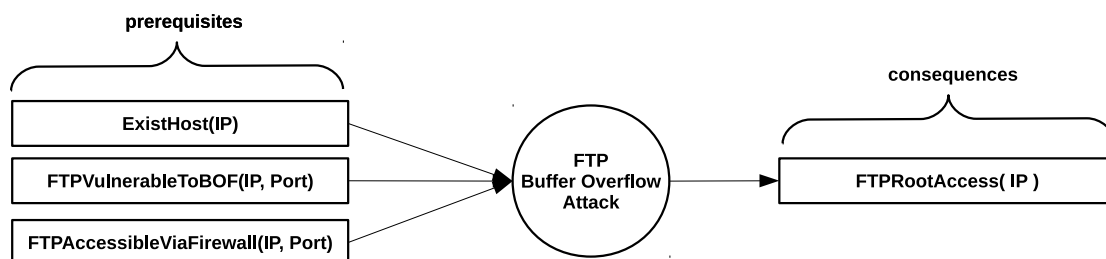


Figure 5.10: Prerequisites and consequences for a buffer overflow attack against a FTP service.

The problem with using prerequisites and consequences to capture causality is the number of conditions that need to be defined for each attack instance, which sometimes can be overwhelming, and the validation of these conditions, which may

not be straightforward. Figure 5.10 illustrates a set of prerequisites and consequences involved in a buffer overflow attack against an FTP service. As we can see from the Figure, the prerequisites and consequences consist of different types of conditions. In Figure 5.10, the condition $FTPAccessibleViaFirewall(IP, Port)$ is related to the network configuration, while the condition $ExistHost(IP, Port)$ is related to the network topology. Defining the prerequisites and consequences of every attack instance with respect to the network topology, the network configuration, and vulnerability information is time consuming. Moreover, validating these conditions is very difficult. For instance, checking whether the FTP service is vulnerable to a buffer overflow attack or not is not an obvious task. This requires using at least some vulnerability scanning tool, and analyzing the scanning results using heuristics.

In our work, we use a different approach to establish the causality between alerts. Specifically, we determine whether two alerts are causally related by considering the impacts of corresponding attacks. The attack impact refers to the outcome of the attack, such as, discovering a host, discovering a service, accessing confidential data, gaining root access, etc. For any two attack instances a and b , if p is an attack impact created by a that is also an enabler for successful execution of b , then there is a causal relationship between a and b . In other words the intruder will execute first a and then b . For instance, the outcome of a scanning attack that detects the presence of an FTP server may enable the execution of a buffer overflow attack against this FTP server (provided of course that the server is also vulnerable to such exploit).

Using the concept of attack impact, we capture the strength of the causal relation between a and b by defining a new causality metric. Let A denote the set of created impacts of attack a and let B denote the set of required impacts for attack b . We define the strength of the causal relation between a and b as a value between 0 and 1 given by equation 5.4.1, where 0 indicates no causality and 1 indicates maximum

causality:

$$causality(a, b) = \frac{|A \cap B|}{|A \cup B|} \quad (5.4.1)$$

One benefit feature of using semantic correlation for reconstructing attack scenario and detecting attack causality is taking advantages of inheritance and semantic similarity. Two attack instances a and b are causally related if their ancestor classes are causally related or if one of them has a semantic similar instance/class that is causally related to the other. For example, if a is semantically similar to c and c is causally related to b (e.g. enabler of b) then we can say that a is causally related to b as well.

The process of detecting attack causality and reconstructing the attack scenario graph can be described as a graph transformation operation. The attack causality detection algorithm converts the complete graph representing the candidate attack scenario into a directed acyclic graph representing the reconstructed attack scenario. The transformation consists of simply replacing the edges in the alerts correlation graph corresponding to the semantic relevance relations between alerts with new edges that represent the causal relations between the attacks reported by the alerts.

To illustrate the transformation of the alert correlation graph into the attack scenario graph let us consider the set of alerts given in Table 5.15. These alerts represent a subset of alerts generated during an attack conducted against our lab honeynet [6]. The alerts were generated using Snort IDS; likewise, the snort signatures triggered by the attack are listed in the third column of Table 5.15. Each of these signatures corresponds to different attack instance. However, they are all semantically relevant because they represent attacks that affect the same service, which in this case is the FTP service.

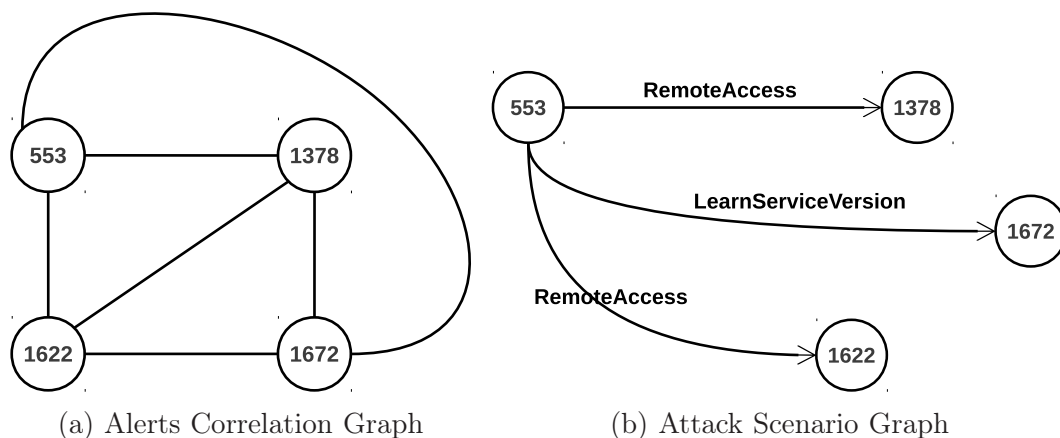


Figure 5.11: Transforming Alerts Correlation Graph (a) to Attack Scenario Graph (b) using the Attack Causality Relation

ID	Source	Target	Snort Attack-Signature
a_1	211.42.48.148	192.168.100.102	1:533
a_2	211.42.48.148	192.168.100.102	1:1622
a_3	211.42.48.148	192.168.100.102	1:1378
a_4	211.42.48.148	192.168.100.102	1:1672

Table 5.15: IDS alerts generated by an FTP vulnerability exploitation attempt.

Figure 5.11-(a) shows the alerts correlation graph for the alerts in Table 5.15. The edges in Figure 5.11-(a) represent the semantic relevance relations between the alerts. Figure 5.11-(b) shows a graph, referred to as an attack scenario graph (ASG), for the alerts in Figure 5.11-(a). The undirected edges in Figure 5.11-(a) are replaced by new directed edges representing causal relations between the attacks in Figure 5.11-(b). The labels on the edges in the ASG in Figure 5.11 -(b) indicate the impact created by the attack instance in node 553 that is required by the attack instances in the other nodes.

Algorithm 6 describes the key steps of the attack causality analysis. The algorithm takes a clique (i.e. a candidate attack scenario) as an input and generates an attack

scenario graph as an output. The input clique is represented by a vector V of alerts sorted in ascending order based on their timestamps. The output of the algorithm is an attack scenario graph represented by a set of matrices denoted M . The algorithm starts by creating an empty matrix m_1 and inserts the first alert in V into m_1 . Then the algorithm iterates $n - 1$ times, where n is the size of V . In each iteration, the algorithm checks the causality between one alert a_i from V and every alert b in every matrix m_j in M using equation 5.4.1. If the causality measure equal zero for every alert in every matrix m_j in M , the algorithm creates a new matrix m_{j+1} and adds a_i to this matrix. If the causality is different from zero then the algorithm will add a_i to the matrix that returns the maximum causality with a_i .

Algorithm 6: Attacks Causality Analysis

```

/* V a sorted vector of alerts that belong to one clique          */
/* M a set of matrices that represent the attack scenario graph  */
/* n number of alerts in V                                       */
/* l number of matrices in M                                     */
Input: V
Output: M
1 begin
2   create m1 as an empty matrix in M;
3   add V[1] to m1;
4   l ← 1;
5   for i ← 2 to n do
6     max ← 0;
7     for j ← 1 to l do
8       foreach alert b ∈ mj do
9         δ ← causality(ai,b);
10        if δ > max then
11          max ← δ ;
12          sMatrix ← mj ;
13          sAlert ← b;
14        end
15      end
16    end
17    if max ≠ 0 then
18      add ai to mj at sAlert;
19    else
20      l ← l + 1;
21      create ml as an empty matrix in M;
22      add ai to ml;
23    end
24  end
25  return M;
26 end

```

The ideal output of the algorithm is the case where M contains a single matrix,

which means that the attack scenario graph is a connected graph. The case where M contains more than one matrix indicates that the attack scenario graph is not a connected graph, which corresponds either to a false negative, a novel attack, or some missing causality information.

5.4.3 Identifying Missing Attacks and False Negatives

To identify missing attacks or potential false negatives, our approach uses in combination the attack causality information and environmental awareness knowledge.

By analyzing an attack scenario graph, we can easily detect missing causality links by identifying disconnected components involved in the graph, which indicate possible missing attacks. It is important, however, to verify that the predicted attacks are valid with regard to the target environment. This means that our assumptions about the missing attacks should match the target system information. For example, a prediction that the missing attack is related to an FTP-exploit will be valid only if the environmental awareness knowledge shows that there is an FTP server running on the target system. This validation is important to avoid the generation of false attack scenarios.

We developed an algorithm with linear time complexity to predict missing attacks. The algorithm takes as inputs two attacks α and δ belonging to two disconnected components in the same attack scenario graph. The output of the algorithm is an attack path p that connects α to δ . The attack path p contains one or more attacks that were most likely missed by the IDS causing the disconnection in the attack scenario graph.

The algorithm defines β as a potentially missed attack between α and δ and set β equal α . Then, using the causality relation it validates one at a time every attack γ that is causally related to β . If the attack γ is valid according to the environmental

Algorithm 7: Predicting Missing Attacks

```

/*  $\alpha$  an attack step in an attack scenario graph */
/*  $\delta$  an attack step in an attack scenario graph where  $\delta \neq \alpha$  */
/*  $\beta$  an attack step connects  $\alpha$  and  $\delta$  */
/*  $\gamma$  an attack step that possibly connect  $\alpha$  and  $\delta$  */
/*  $p$  the set of all attack steps that connect  $\alpha$  and  $\delta$  */
Input:  $\alpha$  and  $\delta$ 
Output:  $p$ 
1 begin
2   Let  $p$  be an empty attack path;
3    $\beta \leftarrow \alpha$ ;
4   while  $\beta \neq \alpha$  or  $\beta$  has causally-related attacks do
5     if  $\beta$  has no causally-related attacks then
6       if  $\beta \neq \alpha$  then
7         remove  $\beta$  from  $p$ ;
8          $\beta \leftarrow$  last attack in  $p$ ;
9       end
10      else
11         $\gamma \leftarrow$  a causally-related attack of  $\beta$ ;
12        remove  $\gamma$  from  $\beta$  causally-related attacks set;
13        add  $\gamma$  to  $p$ ;
14        if  $\gamma = \delta$  then
15          return  $p$ ;
16        end
17        if valid( $\gamma$ ) then
18           $\beta \leftarrow \gamma$ ;
19        end
20      end
21    end
22  return  $p$ ;
23 end

```

awareness knowledge, the algorithm will add β to the attack path p and set γ as the new β and start checking the attacks that directly depend on the new β by repeating the same steps for the new β . If the attack is not valid or the attack has no further causally related attacks, the algorithm will perform a backtracking step. The backtracking step removes the current attack represented by β from the attack path p and set the attack that β depends on (predecessor attack of β) as the new β .

The algorithm will terminate when it reaches the final attack δ where the new β equal δ . In this case it will return the attack path p that contains all the causally related attacks between α and δ . If the backtracking returns to the starting attack α and examines all the attacks that depend on α but never reaches δ , the algorithm will terminate and return the path p equal null. This indicates that the missing attack is either a novel attack or there is a missing causality relation in the knowledge-base.

Another well-known problem in IDS that affect the attack scenario reconstruction is the impact of false positives. False positives occur when a normal behavior is considered by the IDS as malicious, and a false alert is generated as a consequence. Usually, attack scenario reconstruction from a set of alerts involving false positives will generate false attack scenarios or assume the existence of an attack stage that did not really exist.

Our attack scenario reconstruction process does not handle intrinsically false positives. It relies instead on the alert verification component to handle the problem of false positives. This component uses either environmental awareness information or a set of classification rules to eliminate false positives. False positives affect the accuracy of any attack scenario reconstruction technique.

5.5 Summary

In this chapter we tackled the three main tasks in IDS alert analysis, namely alert verification, alert aggregation, and attack scenario reconstruction. We proposed several methods using semantic correlation, ontology and machine learning to overcome the limitations of existing alerts analysis techniques in the literature.

For alert verification, we proposed the use of alert context and semantic analysis to build a robust alert verification technique. We designed two methods for alert verification. The first method uses semantic similarity and KNN to verify alerts and eliminate false positives. The second method is based on a computational model inspired by human immune system and uses ontology and rule induction for alert verification.

For alert aggregation, we proposed an abstraction technique for alert fusion and summarization. The proposed technique relies on using ontology and semantic simi-

larity. We designed two novel methods for alert aggregation. The first method uses hill climbing technique to generalize and fuse the raw alerts based on the intrusion ontology. This method guarantees any desired alert reduction rate. The second method uses semantic similarity to fuse and aggregate raw alerts. The proposed method is effective in detecting information loss and avoids losing important security relevant information during the alert aggregation process.

Finally, we proposed an attack scenario and attack pattern detection and reconstruction technique using semantic-based clustering technique. The proposed technique clusters semantically relevant alerts to build a correlation graph and extracts attack pattern using graph analysis methods. In addition, we proposed a new method to detect causality relation between attacks using attack impact analysis and semantic correlation. Finally, we proposed a new method to detect missing attack steps and mitigate the effect of false negatives in IDS systems.

Chapter 6

Experiments

Finding a dataset to evaluate the proposed alert analysis techniques is a challenge. To fully utilize or demonstrate the benefits of the proposed techniques, we need a dataset that contains multistage computer network attacks. It is important that the dataset is labeled and the attack scenarios are known in advance. Working with unlabeled dataset will not allow us to evaluate the accuracy of our approach. It is also important that the dataset contains attack instances that represent modern attacks and network threats. Finally, it is critical that the dataset contains realistic network traffic. However, to the best of our knowledge all the available benchmark datasets for network intrusions are designed to evaluate intrusion detection systems and not to evaluate IDS alert correlation or post IDS alert analysis. Multistage attacks are not very common in the available benchmark intrusion datasets and most of the labeled datasets contain outdated intrusion instances. In general, it is important to validate any proposed IDS alert correlation technique using different datasets to ensure that the technique does not overfit a particular dataset. For that reason we tested our technique using four different datasets. In this chapter we only discuss the evaluation results with two datasets. However, in our published papers we discussed

the evaluation results with the other datasets [68, 70].

This chapter is organized as follows. First, we introduce the benchmark datasets used in our evaluation. Then, we show the evaluation results and compare our results to the previous work in the literature. Finally, we discuss our results and make some observations.

6.1 Benchmark IDS Datasets

To evaluate our techniques we used two IDS evaluation datasets:

1. UNB ISCX Intrusion Detection Evaluation Dataset: This is one of the most recent intrusion detection evaluation datasets. It was released in 2011 by the ISCX research lab from UNB [76]. It has a realistic network configuration and network traffic. It contains different multistage attacks and modern intrusion instances, such as botnet and DDoS attacks. The dataset contains much larger network traffic compared to other existing benchmark IDS datasets. The dataset was collected over 7 days and it is labeled (marked). The ISCX dataset is the closest dataset to our evaluation requirements. To our knowledge the ISCX IDS dataset has not been used to evaluate any alert correlation or alert analysis technique in the literature. This is because it is a new dataset. We mainly use it to demonstrate how our technique can handle massive number of IDS alerts.
2. DARPA Intrusion Detection Data Sets: Perhaps the DARPA dataset is the most commonly used dataset in the literature to evaluate IDS and alert correlation techniques [41]. In our experiment we are using the DARPA 2000. The DARPA 2000 contains two different simple multistage attacks. It does not really contain realistic network traffic. We evaluate our approach using the DARPA 2000

dataset to compare our results to previous work in the literature, since it is the most commonly used dataset to evaluate alert correlation techniques.

6.2 Evaluation Results

We evaluate how our techniques will perform with a massive number of IDS alerts (alert flooding) as in the ISCX dataset. In particular, we focus on the alert verification and elimination of false positives. In addition, we look at alerts fusion/aggregation and attack scenario construction. Next, we compare our alert fusion and attack scenario reconstruction techniques against the previous works in the literature using the DARPA 2000 dataset. We were not able to compare our alert verification technique to other techniques, because all the alert verification techniques in the literature did not provide enough data for comparison. The network traffic of the three datasets were analyzed using Snort IDS version 2.9.2 to obtain the IDS alerts. We also used Bro IDS to analyze the DARPA dataset in addition to Snort IDS. This additional analysis step allows us to test that how our technique can work in multi-sensor (multiple and different IDSs) environment.

6.2.1 Handling Massive IDS Alerts

Part of our alert analysis framework is a log analyzer that we implemented to understand the characteristics of the generated alerts and to provide basic IDS alerts analysis features. Using our alerts log analyzer we extracted the basic characteristics of the alerts such as the number of alerts in the dataset, the number of hosts, the number of intrusions. Table 6.1 list the properties of the intrusion alerts found by SNORT in the ISCX dataset.

Property	Total Number
raw alerts	339027
intrusion signature	177
IP address	5946
Ports	20858

Table 6.1: ISCX Intrusions Properties

Table 6.2 categorizes the intrusion alerts from the ISCX dataset into true positives and false positives.

Properties	True Positives	False Positives
raw alerts	6972	3,332,055
intrusion signature	34	147
IP address	58	5924
Ports	5578	16927

Table 6.2: Numbers of false positives versus true positives in the ISCX dataset

As we can see from Table 6.2, the number of false positives overwhelms the true positives. Only 2% of the alerts represent true intrusion attempts. We found that 36 out of the 58 IP addresses that were part of true positives intrusions were also linked to false positives.

The above situation illustrates well the alerts flooding problem in intrusion detection systems. Even if we assume that the intrusion analyst will study the alerts on a daily basis, it is highly possible that the analyst will miss the genuine (true positives) alerts. Using our log analyzer we plot the alerts in the ISCX dataset grouped by day and category. As shown in Figure 6.1 the true positives on any day in the dataset are unnoticeable compared to the false positives.

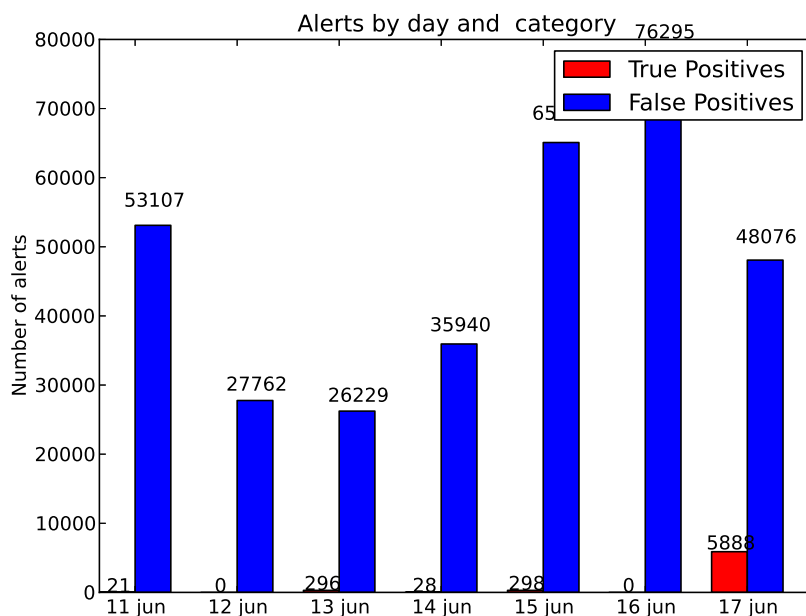


Figure 6.1: ISCX Alert Grouped By Day and Category.

To evaluate our alert verification technique we need to put some constraints in place. This is because the ISCX data set has four different attack scenarios and each scenario is executed only once. Therefore, using this dataset to train any classifier is mostly difficult. In addition, the fact that false positives overwhelm true positives makes it more challenging to learn classification rules. In our evaluation we decide to use a subset of the false positives to train our alert verification techniques. We only use the false positives that share the IP addresses of the true positives. In other words if any host in the network was not part of (attacker/target) a true attack attempt we ignore its alerts, since all of them are known to be false positives. Now if we plot the true positives and the selected subset of false positives, we can see that the false positives still overwhelm the true positives, as shown in Figure 6.2. In our evaluation we have 95,954 false positives and 6972 true positives alerts.

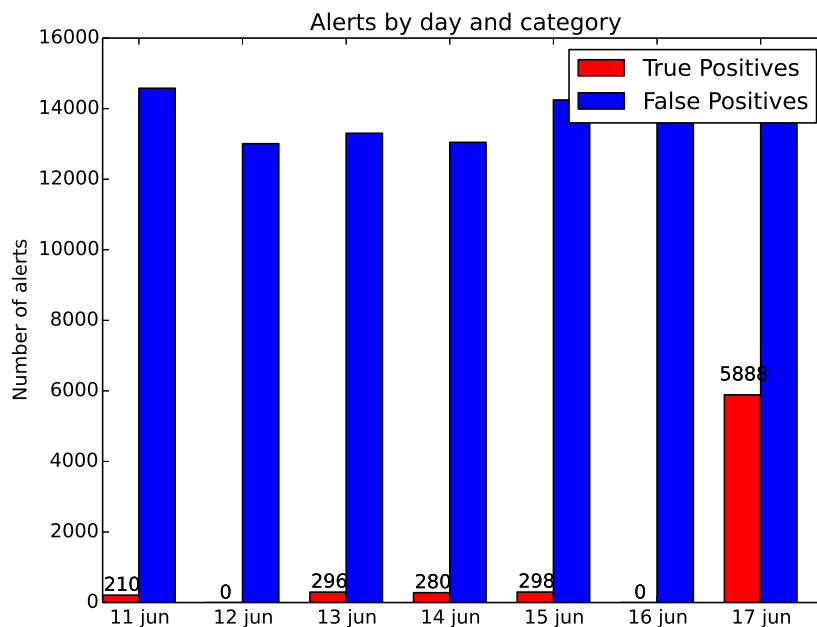


Figure 6.2: ISCX Alert Grouped By Day and Category.

Alert Verification

To evaluate our alert verification technique we divide the IDS alerts into 10 groups where each group has between 18000-21000 IDS alerts. We ensure that each group has both false positives and true positives. Half of the alerts in each group were used for training and the other half for testing. First, we evaluate our alert verification technique that uses KNN (see section 5.2.2), then we evaluate our ontology-based rule induction technique (see section 5.2.3). Table 6.3 shows the results of the alert verification using KNN and our semantic similarity metric. As we can see in Table 6.3 on average our KNN technique reduces **96.58%** of the false positives and on average detects **98.84%** of the true positives alerts. These results were obtained using $k=9$. We noticed a major drop in the detection rate of true positives when we set k greater than **9**. For instance, when we set $k=11$, the average detection rate for true positives was **92.71%**. This is because the amount of false positives is overwhelming compared

to the true positives. Therefore, increasing the number of neighbors we use to classify the alerts is not a good decision. The last column in Table 6.3 shows the time (in minutes) required to classify the alerts. It is clear that the runtime is not suitable for online alert verification.

ID	K1		K3		K5		K7		K9		Time
	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP	
1	83.63	92.49	92.85	94.55	96.52	95.38	98.85	95.90	98.88	95.90	70.56
2	81.70	88.79	84.90	89.59	95.94	92.36	97.81	92.83	97.84	92.84	67.24
3	83.52	91.42	96.81	94.68	99.29	95.29	99.29	95.29	99.29	95.29	109.71
4	83.60	90.36	96.80	93.66	97.69	93.88	99.28	94.28	99.28	94.28	60.29
5	82.91	94.45	97.02	98.14	98.42	98.92	98.42	98.92	98.42	98.92	83.36
6	84.55	92.50	86.33	92.91	96.00	95.18	97.20	95.46	98.95	95.87	62.93
7	83.69	89.28	84.70	89.52	92.17	91.28	96.71	92.35	98.30	92.72	64.77
8	83.49	84.72	95.60	93.94	96.42	97.61	97.33	99.94	98.73	99.97	76.89
9	85.57	96.58	87.35	98.64	94.03	99.46	98.22	99.99	99.97	99.99	90.48
10	85.36	96.55	86.37	96.79	93.84	98.55	97.38	99.62	98.72	99.99	78.28
avg	83.80	91.71	90.87	94.24	96.03	95.79	98.05	96.46	98.84	96.58	76.45
min	83.63	84.72	86.37	89.52	93.84	91.28	97.38	92.35	98.72	92.72	60.29
max	85.57	96.58	97.02	98.64	99.29	99.46	99.29	99.99	99.97	99.99	109.71

Table 6.3: Alert Verification Using KNN and Semantic Similarity

Using the same 10 groups of IDS alerts and the same settings we evaluate our ontology-based rule induction technique for IDS alert verification. Table 6.4 shows the results of the alert verification using our ontology-based rule induction technique. As we can see using rule induction on average we were able to reduce **98.21%** of the false positives and detect **98.43%** of the the true positives. One obvious advantage of the rule induction over the KNN is the runtime. While the (learning) time required to generate the rules is close to the time of the KNN , we can see in Table 6.4 that the matching time for the rule induction is about 7 times less than the time required

for the KNN. Therefore, we think that a rule induction approach is suitable for online alert verification.

While our rule induction technique has a better on average detection rate for both false positives and true positives alerts we do not claim that the rule induction in general is better than the KNN for alert verification. After all, the results in Tables 6.3 and 6.4 are only observed from a single evaluation run on a single dataset.

Finally, an interesting observation we found in the results is about the botnet attack scenario in the dataset. This attack scenario contains an IRC bot: the C&C communications of the bot are detected by Snort as normal IRC traffic (Snort consider it a possible site policy violation) not as bot C&C or malicious IRC. Since the dataset contains massive amount of IRC traffic that are not labeled malicious (i.e. not related to the IRC bot), Snort fires the same alert for both the malicious IRC (bot related) traffic and the none malicious IRC traffic. However, our techniques (KNN and OBRI) were able to distinguish between the false positive and the true positive alerts. This is mainly because the use of alert context features and semantic analysis.

ID	Rule Induction		Detection Accuracy		Time	
	Gene Library	Rules	TP	FP	Learning	Matching
1	182.00	481.00	97.69	98.63	55.08	9.76
2	173.00	473.00	98.62	98.31	51.80	7.31
3	282.00	499.00	98.15	98.84	94.29	15.51
4	155.00	472.00	97.38	95.68	44.87	9.19
5	214.00	483.00	98.62	96.98	67.96	10.11
6	162.00	470.00	97.69	99.04	47.48	9.78
7	215.00	478.00	98.77	98.70	67.98	13.03
8	198.00	480.00	98.97	97.18	61.32	10.66
9	225.00	483.00	99.29	99.37	72.16	12.80
10	198.00	473.00	99.15	99.41	61.48	15.06
avg	200.40	479.20	98.43	98.21	62.44	11.31
min	155.00	470.00	97.38	95.68	44.87	7.31
max	282.00	499.00	99.29	99.41	94.29	15.51

Table 6.4: Alerts Verification Using Ontology and Rule Induction

Alert Aggregation

To test our alert aggregation technique we used the set of true positives alerts, which contains 6972 IDS alerts. Again, each raw IDS alert is processed and described using seven attributes, namely, alert source, alert destination, attack class, attack impact, affected service, affected application, and affected platform. In the first part of our experiment we tested our lightweight alert aggregation technique that uses a hill climbing approach. In this part we set the desired number of hybrid alerts between 130 to 150 hybrid alerts. In other words we represent the 6972 raw IDS alerts using only between 130-150 hybrid alert which will give us an alert reduction rate (ARR) between [97%-98%]. We run our experiment seven times, where each time a different alert attribute is used to start the generalization process. Table 6.5 shows the results

of our experiment. Note that the average runtime of each round of the seven rounds in our experiment was close to 1 minute.

ID	Alerts Reduction Rate (ARR)	information Loss Rate (ILR)		
		minimum loss	average loss	maximum loss
1	97.83	27.31	40.66	55.98
2	97.87	26.90	40.40	53.40
3	97.99	24.66	40.59	53.40
4	97.84	24.98	40.98	54.64
5	97.80	26.29	40.39	54.79
6	97.75	27.76	40.66	55.98
7	97.86	24.29	40.56	54.64
min	97.75	24.29	40.39	53.40
avg	97.85	26.03	40.61	54.69
max	97.86	27.31	40.66	55.98

Table 6.5: A Lightweight Alerts Aggregation Using Hill Climbing Approach

As we can see from Table 6.5 our algorithm was able to fuse the raw alerts to reach the desired number of hybrid alerts. For each of the seven rounds, we calculate the information loss rate (ILR) for each generated hybrid alert. It is clear that while the algorithm can reach the desired alert reduction rate, it is possible to generate hybrid alerts with high information loss rate. For example the hybrid alert with the maximum information loss in our experiment was generated in round six, with an ILR equal 55.98%. This means more than 55% of the information in the raw alerts are missed by representing them using this hybrid alert. Here is the hybrid alert with the maximum information loss:

```
{ Source: North_America, Destination: Internal_Network, Attack: Implementation_Bug,
Impact: System_Compromise, Application: Web_App, Service: Web_Server, OS: Windows }
```

In our opinion judging the quality or the usefulness of the above hybrid alert is up to the intrusion analyst and the intended use of the aggregation results.

In the second part of our experiment with alert aggregation we aggregate the alerts based on their semantic similarity using the algorithm proposed in Section 5.3.2. Table 6.6 shows the results of aggregating the alerts based on their semantic similarity. The results in Table 6.6 are obtained with semantic similarity threshold equal **0.8**. This means only alerts with semantic similarity above **0.8** can be aggregated/fused into one hybrid alert. Based on experiments conducted using other datasets (see section 6.2.2) a semantic similarity threshold less than **0.7** or greater than **0.9** most likely will give poor aggregation results, by either yielding very high information loss or failing to aggregate the majority of the raw alerts. We discuss threshold selection in connection to one such experiments later using the DARPA dataset.

ID	Alerts Reduction Rate (ARR)	Information Loss Rate (ILR)		
		minimum loss	average loss	maximum loss
1	92.84	0.00	1.86	9.41
2	94.00	0.00	2.19	11.71
3	92.07	0.00	1.49	7.93
4	91.43	0.00	1.13	6.97
5	94.43	0.00	2.36	12.76
6	93.40	0.00	2.05	10.46
7	93.78	0.00	2.18	11.69
min	91.43	0.00	1.13	6.97
avg	93.13	0.00	1.89	10.13
max	94.43	0.00	2.36	12.76

Table 6.6: Alerts Aggregation Based on Alerts Semantic Similarity

From Table 6.6 we can see that aggregating IDS alerts based on their semantic similarity improves the information loss rate and reduces on average 93% of the raw IDS alerts. Note that for all the rounds of our experiment the minimum ILR equal **0.00**; this is because some alerts will not be aggregated at all. These alerts were not fused or generalized with any other alerts in the raw alerts set because their semantic similarity does not meet the predefined threshold. It is clear from the results in

Table 6.6 that when the alert reduction increases the information loss increases as well. However, it is possible that the alert reduction increases without increasing the information loss rate. For instance, aggregating a group of identical alerts can give us 99.0% alert reduction rate and zero information loss rate. The maximum information loss rate (12.76%) is recored in round 5. Here is the hybrid alert with the maximum information loss:

```
{ Source: Internal_Sub_Network_2, Destination: Internal_Network, Attack:
Abnormal_IRC_Traffic, Impact: Policy_Violation, Application: IRC_Client, Service: IRC ,
OS: Windows }
```

Attack Patterns and Scenario Extraction

Here we evaluate the ability of our proposed technique to extract attack patterns and reconstruct attack scenarios from a set of true positive alerts. In the ISCX dataset there are four main attack scenarios, each consisting of several attack steps. The attack steps of each attack scenario can be grouped into two or more attack patterns.

At the beginning, we use our technique to build the alert correlation graph for each attack scenario. Then, from each alert correlation graph we extract attack patterns and apply attack causality analysis to reconstruct the attack scenario.

The first attack scenario in the ISCX dataset is titled "Infiltrating the network from the inside". This attack scenario consists of 7 attack stages. Figure 6.3 shows the alerts correlation graph of this attack scenario. As we can see from Figure 6.3 the ACG is a disconnected graph and has 3 maximal cliques. Here we use colors and node numbers to improve the visualization of the graph and easily distinguish between the different cliques (candidate attack stages/patterns).

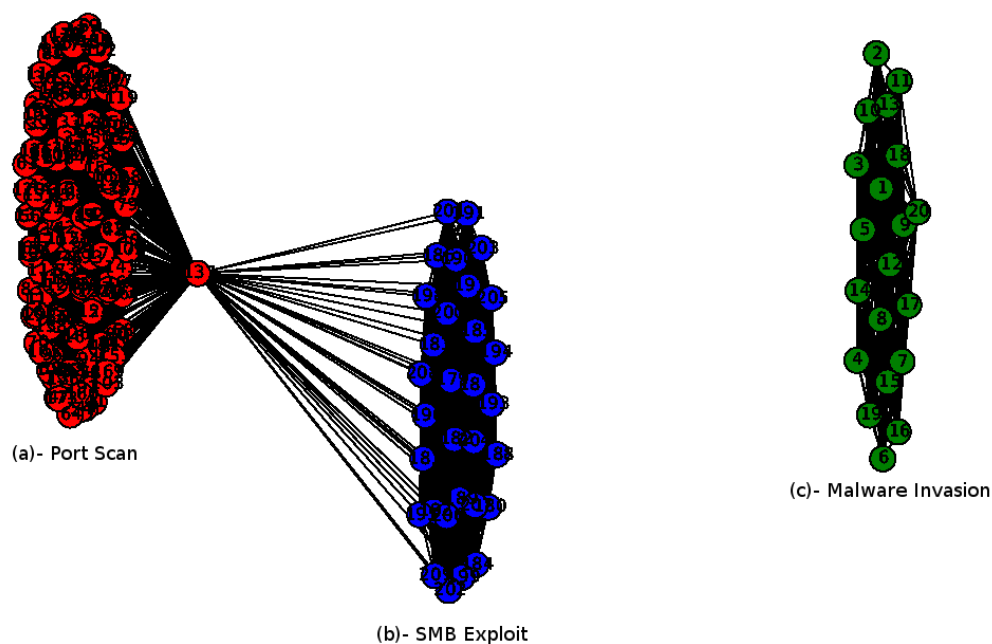


Figure 6.3: Alert Correlation Graph for the first Attack Scenario from the ISCX dataset

The vertices in Figure 6.3-(c), labeled from 0 to 21 represent one attack stage which is *malware invasion*. The vertices in Figure 6.3-(a) labeled from 22-178 represent the second attack stage which is *port scan*. Finally, the vertices in Figure 6.3-(b), labeled from 179 to 210 represent the third attack scenario which is *SMB exploit*. We can see that vertex 137 is shared between two cliques/attack stages, the port scan and the SMB exploit. However, our algorithm clusters this vertex with the port scan attack not the SMB exploit. This is because the total semantic relevance between the alert represented by vertex 137 and the alerts in the port scan attack is greater than the semantic relevance between this alert and the SMB attack. In fact, the alert represented by vertex 137 only shares the destination with the alerts in the SMB attack.

By applying our attack causality analysis algorithm on the attack stages in the ACG we were able to construct the attack scenario graph as illustrated in Figure 6.4. Based on the constructed attack scenario graph, we can describe the scenario corresponding to the "Infiltrating the network from the inside" attack. The attack begins with a malware invasion from the Internet to subnets 1, 2, and 4 inside the ISCX network. One host from subnet 1 was successfully infected by the malware and started a port scanning attack against subnets 1 and 2. Then this host executed an SMB buffer overflow exploit against one of the hosts in subnet 2 that was discovered during the port scan attack (previous attack stage).



Figure 6.4: Attack Scenario Graph for the First Attack Scenario from the ISCX dataset

We mentioned before that the "Infiltrating the network from the inside" attack scenario consists of 7 attack stages, as stated in the ISCX dataset documentation. However, here our technique only constructs an attack scenario consisting of 3 stages. Therefore, our technique successfully constructs part of the actual attack scenario.

In other words, the attack scenario constructed here is incomplete. We try to understand why we end up with an incomplete attack scenario. In particular, we want to determine whether or not this a limitation in our technique.

Our technique missed the following four attack stages: external network probe, Adobe Reader exploit, SQL Injection and backdoor installation. The external network probe occurred before the malware invasion, while Adobe Reader exploit occurred after the malware invasion and before the port scan. The SQL Injection and the backdoor installation both occurred after the SMB exploit. Now, the network probe was a passive information gathering attack (e.g. collecting information on the target using search engine), where the attacker did not directly interact with the ISCX network to gather intelligence about the network. Therefore, this attack stage cannot be detected by the IDS and as a result it cannot be reconstructed by our technique. The Adobe Reader exploit cannot be detected by the network IDS since it is executed on a victim host and not over the network. However, it can be detected using a host-based IDS. Therefore, if relevant host-based IDS alerts log is provided, our technique should be able to reconstruct this stage. The remaining stages can be detected by NIDS and other security appliances such as web-application proxies. However, Snort did not detect these attack stages, because the default rule-set and attack signature that come with Snort are not build to detect insider attacks. Since, none of the missing attack stages were detected by the IDS, it is not possible for our technique to reconstruct these missing attack stages.

The second attack scenario in the ISCX dataset is "HTTP Denial of Services". The attacker used the back-door installed in the first scenario to gain access to the network. Then, he exploited an SMB vulnerability on a host in subnet 3. Then, through a a connection started from the ISCX network, he downloaded the famous malicious HTTP DOS tool "Slowloris". Slowloris was installed inside the ISCX network and

used to attack the main web server in subnet 5. Figure 6.5 shows the Attack Scenario Graph of the second attack scenario "HTTP Denial of Services".

Snort detects the "Slowloris" download as a binary/exe file download. This triggers Snort to generate abnormal traffic alerts to describe this download action. We had to write our own Snort rules to enable Snort to detect Slowloris DOS attack. This is because the default Snort rule-set does not come with Slowloris signature.



Figure 6.5: Attack Scenario Graph for ISCX Second Attack Scenario

The third attack scenario is a "Distributed Denial of Service using an IRC Botnet". Our attack scenario reconstruction technique was able only to reconstruct the invasion of the ISCX by the IRC bot client/server, and the abnormal IRC traffic. The reconstructed attack scenario missed the final stage where the bots execute a DDOS attack against the ISXC web server. This is mainly because the IDS fails to detect the bot attack behaviors. Figure 6.6 shows the reconstructed attack scenario graph

of this "Distributed Denial of Service using an IRC Botnet" attack.

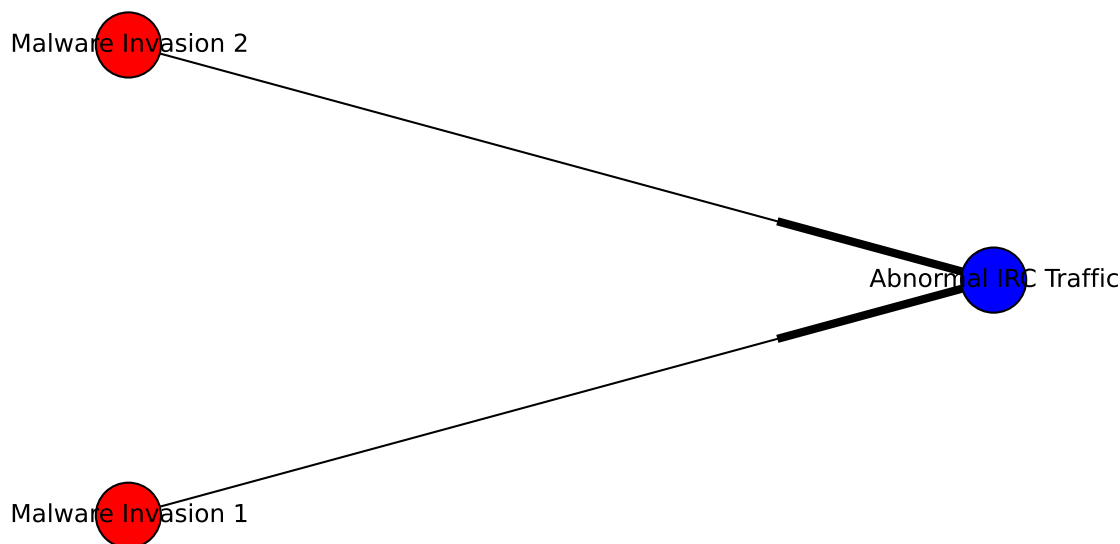


Figure 6.6: Attack Scenario Graph for ISCX Third Attack Scenario

The fourth and last attack scenario in the ISCX dataset is "Brute Force SSH" attack that executes a password dictionary attack against an ssh account in the ISCX network. There is not much details about this attack scenario in the description of the ISCX dataset. However, by analyzing the result of our alert analysis techniques we can tell that the attack consists of one stage which is the dictionary attack.

What is interesting about the "Brute Force SSH" attack in our opinion is the alert correlation graph. After constructing the alert correlation graph we discovered that it consists of 5888 node/vertex where each vertex represents one raw IDS alert. None of these alerts has any causal relation with the other alert in the graph. This indicates that they all belong to a single attack pattern/stage and that the raw alerts share a high semantic similarity. When we try to visualize (draw) the alert correlation graph, our tool (the code invokes graphviz to visualize graph) crashes with out of memory exception. So, we thought it might be useful instead of constructing the alert correlation graph using raw IDS alerts, we could use the hybrid alerts that result

from the alert aggregation. Figure 6.6 shows the alert correlation graph constructed based on the hybrid alerts instead of the raw alerts.

Using the hybrid alerts result from the alert aggregation phase not only makes it possible to visualize the alert correlation graph but it improves the visualization of the graph. We summarize in the following the information extracted from the alert correlation graph. Each vertex in the graph represents a hybrid alert and is labeled by the source of the hybrid alerts. All the hybrid alerts share the same destination which is the victim of the ssh brute force attack. The size of the vertex indicates the number of raw alerts generated by the IDS and fused in one hybrid alert.

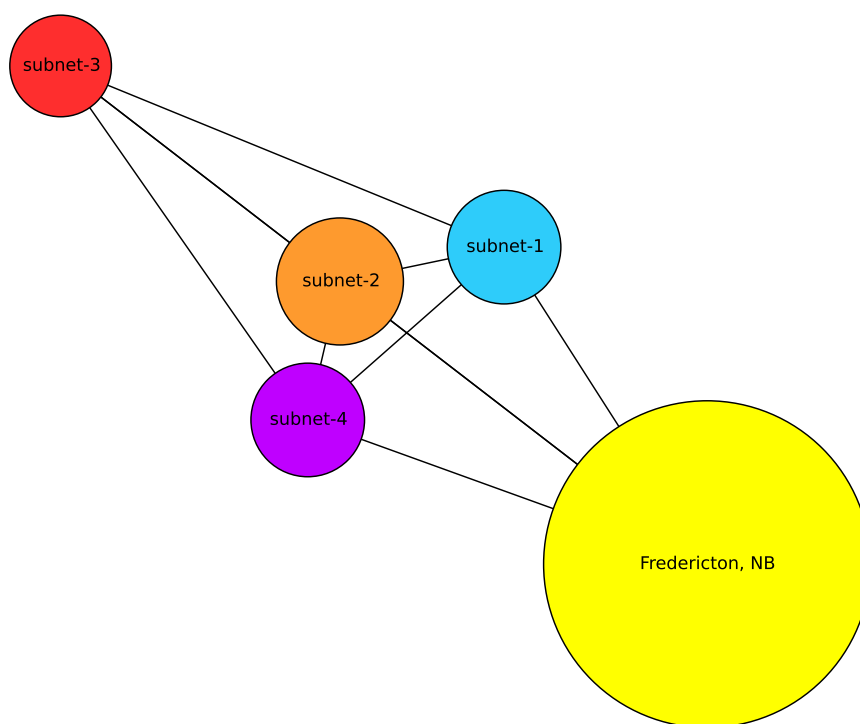


Figure 6.7: Alert Correlation Graph for ISCX Fourth Attack Scenario

As, we can see more than 5000 raw IDS alerts can be represented by only 5 hybrid alerts. In addition, from the ACG we can see that the vertex "Fredericton, NB" has a larger number of alerts compared to the other vertices. Therefore, it is clear that

the source represented by this vertex played the main role in the "SSH Brute Force" attack.

6.2.2 Performance Comparison Using DARPA IDS Dataset

Because the ISCX dataset is new dataset, it is not yet commonly used in evaluating alert correlation techniques. We found that the majority of the alert correlation techniques in the literature use the DARPA 2000 dataset to evaluate their performance. For this reason and for the sake of comparison we test our techniques with the DARPA 2000 dataset and compare our results with those obtained in the literature. However, in our the DARPA 2000 dataset is not the best dataset to evaluate intrusion alert correlation techniques. This is because the dataset contains two basic network attack scenarios, that can easily be recognized by an intrusion analyst and it does not contains realistic network traffic. In addition, the DARPA 2000 dataset, shares most of the limitations of its predecessor datasets DARPA 98 and DARPA 99 [46].

We will use the DARPA dataset in evaluating our alert aggregation/fusion and attack scenario reconstruction technique. By running Snort on the DARPA dataset, only few false positive alerts are found, which is not useful in testing our alert verification technique. This is mainly because the DARPA dataset is very old (created in 2000) and today Snort detection engine and rule-set are well written and should not fail to detect old attacks such as the ones involved in the DARPA dataset. Table 6.7 shows some statistics about the DARPA dataset after analyzing it with Snort such as the number of raw alerts, number of hosts, durations and numbers of different intrusion instances.

Properties	Value
raw alerts	2170
intrusion signature	16
IP address	1055
Duration	≈ 100 min

Table 6.7: DARPA 2000 DOS1.0 Dataset Statistics

Alert Aggregation

We run our semantic-based alert aggregation algorithm 10 times, using each time a different semantic similarity threshold vector. Here we are investigating the relation between the alert reduction rate and the information loss rate with respect to the semantic similarity threshold. Each time, we calculate the alerts reduction rate and the maximum information loss rate. In addition we try to test the performance of the alert aggregation when the raw alerts are generated by a single sensor (IDS) and when it is generated by multiple sensors (IDS from different vendors).

In our evaluation, we used (without loss of generality) three different symbolic attributes to represent IDS alerts, namely, the *attack source*, the *attack target*, and the *intrusion type*. Note that several other attributes can be added to this list such as *attack time*.

We considered for each attribute five different semantic similarity threshold values between zero and one. Using the threshold values we can generate up to 125 different threshold vectors which correspond to all possible combinations of the selected values. In our experiment, we used a subset of 10 different threshold vectors listed in Table 6.8.

Vector ID	Source	Target	Intrusion Type
V0	1	1	1
V1	0.8	0.7	1
V2	0.8	0.8	0.9
V3	0.8	0.8	0.8
V4	0.8	0.7	0.8
V5	0.6	0.7	0.8
V6	0.6	0.6	0.8
V7	0.6	0.6	0.75
V8	0.4	0.4	0.45
V9	0.16	0.16	0.12

Table 6.8: Semantic Similarity Threshold Vectors

Table 6.9 shows the results of our experiment with the single sensor alert aggregation. We plot for each dataset what we refer to as the *Aggregation Performance Curve (APC)*, which shows the relation between the alert reduction rate and the information loss rate when the threshold values vary.

Threshold Vector	ARR	ILR
V0	32.00	0.00
V1	32.00	0.00
V2	83.00	16.00
V3	91.00	17.00
V4	92.00	28.00
V5	99.00	32.00
V6	99.00	32.00
V7	99.00	32.00
V8	99.00	63.00
V9	99.00	91.00

Table 6.9: DARPA Semantic-based Alert Aggregation Results

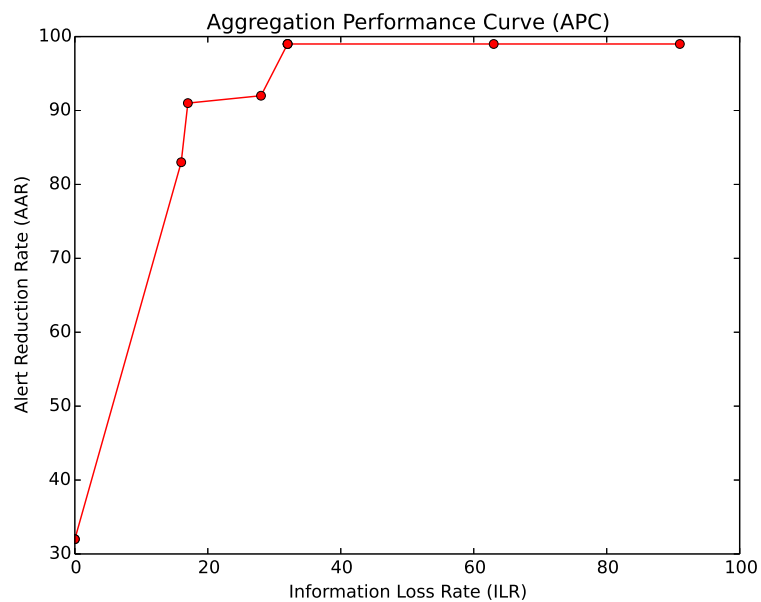


Figure 6.8: APCs for single sensor IDS alerts aggregation

By analyzing the results we find that in general higher alerts reduction rate means

higher information loss rate. We also find that changing the semantic similarity threshold vector will result in one of the following three outcomes. The first outcome is a notable change in the *ARR* and the *ILLR*; this occurs, for instance, when the threshold vector changes from *V2* to *V3*. As we can see there is a notable change in the *ARR* and *ILLR*. The second outcome is no change at all in the *ARR* and *ILLR*; for instance, this is the case when we change the thresholds from *V1* to *V2* or from *V6* to *V7*. The reason for that is because the semantic similarity values between the alerts are less than the semantic similarity threshold values. The third outcome is a notable change in the *ILLR* while the *ARR* barely changes. For instance, as we can see from Table 6.9, changing from *V7* to *V8* or *V9* does not cause any notable change in the *ARR*, however, there is a major change in the *ILLR*.

Also the attack pattern has a significant impact on the alerts reduction rate, the information loss rate and the selection of the semantic similarity threshold. For example, we found that different attack patterns require different adjustments of the semantic similarity threshold for each alert attribute. For instance, 38.4% of the DARPA 2000 raw IDS alerts are related to the Mstream DDoS attack where all the alerts have spoofed, random source IP addresses. Snort uses 2 intrusion signatures to represent that DDoS attack pattern. Since the source IP address in the alerts are spoofed and random we had to set the semantic similarity of the source attribute to lower value to be able to aggregate the alerts that belong to that attack pattern. In fact several existing works in the literature set the similarity thresholds between alert attributes based on the intrusion pattern [25] or define a set of rules to aggregate the alerts based on the type of attack pattern (see for instance, [84, 19]).

High value of alerts reduction rate should not be considered the main factor to judge the performance of any alerts aggregation approach. An intrusion analyst should also consider the amount of information loss in the generated hybrid alerts.

In our experiment we found that the acceptable level of information loss rate can be defined based on the attack pattern. For instance, when aggregating the alerts of the DDoS attack in the DARPA 2000 dataset we obtained an information loss rate of 32%. This rate is mainly the result of aggregating alerts with different source IP addresses. However, these IP addresses are randomly spoofed by the Mstream worm. In this case, the 32% of information loss is acceptable because the randomly spoofed IP addresses do not really bring any useful knowledge to the intrusion analyst. In general, we found that when applying the same semantic similarity threshold to the same intrusion pattern (e.g. DDoS attack, scan attack, etc) in different datasets we obtain the same performance rates (i.e. same ARR and ILR values).

Figure 6.9 depicts the hybrid alert generated for the DDoS attack in the DARPA 2000 dataset from the aggregation process. The source of the hybrid alert is an aggregate attribute value that represents a set of IP addresses that belong to the local network. The target of the attack is an off-site IP address. The intrusion type is Mstream-DDoS, which is also an aggregation of the two original Snort signatures in the raw IDS alerts.

$$HA = \left\{ \begin{array}{ccc} Source & Target & Intrusion \\ 127.X.X.X & 131.84.1.31 & mstream \\ & & DDoS \end{array} \right\}$$

Figure 6.9: Hybrid alert obtained from the DARPA 2000 dataset; the hybrid alert represents a mstream DDoS Attack.

As indicated earlier, the DARPA dataset is the most widely used dataset for the evaluation of alert aggregation approaches. Table 6.10 shows a comparison between our approach and previous approaches from the literature that used the DARPA 2000 dataset.

It is important to point out that the approaches proposed in the literature used either different IDS sensors or IDS rulesets to generate the alerts from the DARPA dataset. This means Table 6.10 does not give a fully accurate comparison. Another important point is that none of the existing multi-sensor aggregation approaches have used the DARPA 2000 dataset in their evaluation. Likewise, all the approaches listed in Table 6.10 are single sensor ones. Also none of the existing approaches provided explicitly the ILR measure. The only existing approaches for which the ILR can be inferred are the ones that use perfect match to aggregate the alerts; in this case the information loss rate is always zero. The attack thread reconstruction approach proposed in [20] fits under this category and yields ($ARR = 6.61\%$, $ILR = 0\%$). Our approach achieves $ILR = 0\%$ when the semantic similarity threshold vector is set to ones as shown by the case of vector V_0 in Table 6.8, which is also a case of perfect match. It must be noted that approaches based on perfect match can only aggregate duplicated alerts, in which case the ARR will depend on the number of duplicated alerts available in the dataset.

Approach	Reference	ARR	ILR
Xu et al.	[92]	64.20%	not measured
Hofmann and Sick	[30]	99.00%	not measured
Wen et al	[88]	91.00%	not measured
attack thread recon	[85]	6.61%	0%
attack focus recognition	[85]	49.58%	not measured
Zhuang et al	[96, 90]	98.70%	not measured
Jie et al	[44]	90.00%	not measured
Our approach	Current	99.0%	32%

Table 6.10: Comparison of alerts aggregation approaches using the DARPA 2000 dataset in their evaluation.

In order to evaluate our approach for multi-sensor IDS alerts aggregation, in addition to the Snort IDS, we used Bro IDS to analyze the network traffic and generate the IDS alerts. In the multi sensor experiment we only used the DARPA 2000 dataset.

As mentioned earlier, the DARPA 2000 dataset contains one attack pattern which is a multi-steps DDoS attack. The intruder probed the network and exploited a Solaris OS services vulnerability to gain root access. He then installed a malware on 3 machines and then used the malware via telnet to attack a remote site. The malware executed a *mstream DoS* attack.

Using our aggregation tool, the raw alerts generated by Snort and Bro were pre-processed and reformatted to provide unified alerts messages based on our ontology. Each attack step was reported in Snort and Bro by one or more attack signatures. Table 6.11 shows the number of raw alerts and intrusions generated by Snort and Bro based on the DARPA dataset and the number of alerts and intrusion after preprocessing the alerts messages and reformatting them based on the ontology vocabularies.

Bro IDS detected five different attack types and generated 880 corresponding raw IDS alerts when analyzing the DARPA dataset. Bro IDS, however, failed to detect the final step of the attack, which is the *mstream DoS*; this was the only step that was not detected by Bro. From Table 6.11 we can notice that after preprocessing Snort and Bro alert messages the number of intrusions increases to 18. This is because only 3 intrusions were commonly detected by both Snort and Bro.

Alerts Format	Alerts Count	Intrusions
Snort	2170	16
Bro	880	5
Ontology-Based	3094	18

Table 6.11: DARPA dataset preprocessing statistics

Table 6.12 shows the results of our experiment for multi-sensor alert aggregation, when varying the thresholds; Figure 6.10 illustrates the corresponding APC. The results of aggregating the DARPA dataset alerts generated by Snort and Bro are very close to the results of aggregating the alerts generated by Snort only. The main reason for that is because Bro failed to detect the mstream DDoS attack.

Vector	DARPA	
	ARR	ILR
V0	0.37	0
V1	0.37	0
V2	0.87	0.16
V3	0.94	0.17
V4	0.99	0.32
V5	0.99	0.32
V6	0.99	0.32
V7	0.99	0.32
V8	0.99	0.63
V9	0.99	0.91

Table 6.12: Multi-sensor Alerts Aggregation Evaluation Results

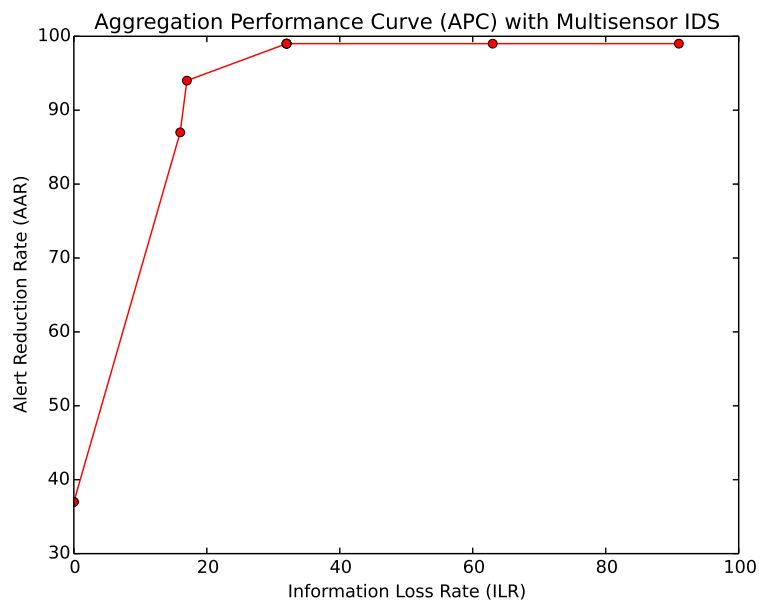


Figure 6.10: APCs for single sensor IDS alerts aggregation

Using $V4$ as threshold vector allows us to achieve ($ARR = 99\%$, $ILR = 32\%$). As mentioned above, $ILR = 32\%$ can be considered acceptable because the loss is related to spoofed IP addresses, which do not bring any useful information.

It is important to emphasize that none of the existing multi-sensor aggregation approaches from the literature have actually been evaluated experimentally in true multi-sensor settings. While the proposed approaches were presented as being able to aggregate multi-sensor alerts, experimental results have been provided only for single-sensor alerts. No quantitative performance results were provided for multi-sensor alerts, which make it difficult to compare objectively our approach against these approaches.

Attack Patterns and Attack Scenario Extraction (AKA Alert Correlation)

We applied our attack scenario reconstruction approach to the 2170 alerts generated by Snort for the DARPA dataset. We obtained at the end of the process, an alert cor-

relation graph consisting of two disconnected subgraphs, containing 1333 nodes and 837 nodes, respectively. The first subgraph involves four cliques, each containing a set of nodes (alerts) representing one attack stage. Likewise, the first subgraph yields the first four attack stages consisting of *Ping Sweep*, *Service Probe*, *Buffer Overflow*, and *Privilege Escalation*. Figures 6.11 and 6.12 shows the alert correlation graph of the *Ping Sweep* and *Privilege Escalation* attack stages.

The second subgraph contains three cliques corresponding to the following attack stages: *Flood Attack*, *Protocol Exploit*, and *Host Probe*.

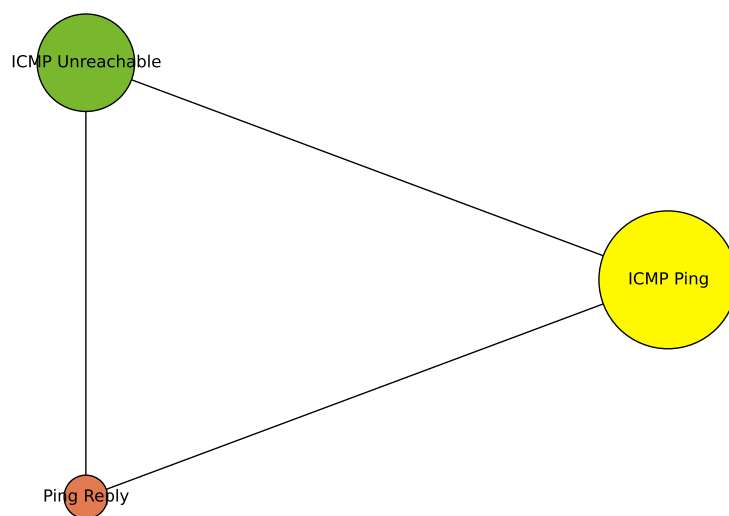


Figure 6.11: Ping Sweep Alerts Clique

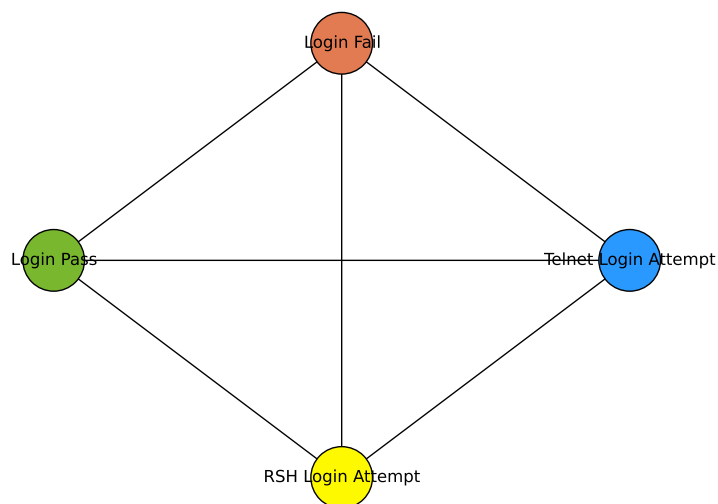


Figure 6.12: Privilege Escalation Alerts Clique

Our approach allows successfully finding causal relations between the four cliques from the first subgraph as shown in Figure 6.13. The attack scenario graph in Figure 6.13 shows the attacks and the underlying causal relationships. The arcs in Figure 6.13 depict causal relations and are labeled by the attack impacts used to detect the causality.

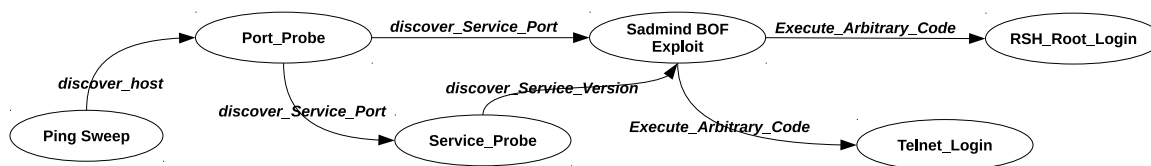


Figure 6.13: Reconstructed Attack Scenario Graph

However, no causality relation was detected between the other three attack stages from the second subgraph (i.e. *Flood Attack*, *Protocol Exploit*, and *Host Probe*).

Since the LLDDOS1.0 dataset contains exactly one attack scenario, the gap between the two subgraphs certainly corresponds to missing attacks (i.e. false negatives) that the Snort IDS failed to detect. Furthermore, the alerts that belong to *Proto-*

col Exploit and *Host Probe* are false positives (generated by the Snort IDS) and thereby irrelevant. This is consistent with the fact that no causality relation was found by our approach between the elements of the second subgraph.

We removed the false positives (related to *Protocol Exploit* and *Host Probe*), and rerun our algorithms in order to predict the missing attacks (false negatives). Using our tool we detect several paths that link the disconnected subgraphs. Here our tool cannot recommend one path over the others due to the lack of environmental information. For that reason we enabled all Snort rules related to the predicted attack paths and reanalyzed the LLDDOS1.0 dataset with Snort. The MStream DDOS ruleset from Snort caused the generation of 26 new alerts containing three unique Snort intrusion signatures, namely, 243, 244, and 246. These alerts indicate communication patterns of MStream DDOS attack. Hence, the MStream infection and the MStream DDOS attack were the missing attacks steps.

Table 6.13 shows our evaluation performance for the LLDDOS1.0 dataset using the soundness and completeness metrics. The table also summarizes the results obtained by other approaches that used the LLDDOS1.0 dataset.

Approach	Completeness	Soundness
Ning et al [52]	93.96%	93.96%
Liu et al [42]	87.12%	86.27%
Al-Mamory and Zhang [3]	86.5%	100%
Li et al [40, 38, 79]	92.2%	not provided
Batani et al [12]	94.1%	95.0%
Our Approach	97.3%	99.70%

Table 6.13: Comparison of Attack Scenario Reconstruction Approaches Using the LLDDOS1.0 Dataset

As shown by Tables 6.13 our approach outperforms many of the previous approaches. The completeness of our approach is promising and shows that our approach can correlate alerts that belong to the same attack scenario with high detection rate. At the same time the soundness of our approach is in general better than most of the previous approaches. The main reason our approach for alert correlation yields higher completeness and soundness in comparison to previous approach is the semantic clustering based on semantic relevance tends to maximize the number of correlated alerts, which increases the number of correlated alerts over the total number of related alerts. The clique analysis tends to group only highly similar alerts into optimum number of clusters, which increases the number of correctly correlated alerts over the total number of correlated alerts.

6.3 Summary

In this chapter we evaluated our semantic-based alert correlation approach using different datasets to show the validity of our proposed approach. Our experiment shows that the proposed techniques can effectively handle massive number of IDS alerts and complex multistage attacks. In addition, the performance of our approach in comparison to previous approaches in the literature is very promising.

Our experiment with alert verification shows that both the KNN algorithm with semantic similarity metric and the ontology-based rule induction algorithm are efficient techniques for detecting false positives with a detection accuracy above 96.00%. The use of semantic similarity and ontological structure to aggregate and fuse IDS alerts show an outstanding alert reduction rate that can reach 99.00%. In addition, we take into account the information loss rate, which is an important factor in evaluating any alert fusion or aggregation technique. Reconstructing a multistage attack

and discovering the attack scenario is the most challenging task in IDS alert correlation. Our technique that is based on semantic analysis and semantic correlation was able to reconstruct attack scenario either completely or partially. In addition, we were able to partially tolerate false negatives in IDS.

NOTE: *Our intrusion ontology can be downloaded from the ISOT lab web site:*

<http://www.uvic.ca/engineering/ece/isot/datasets/index.php>

Chapter 7

Conclusion

Nowadays, with the rapid size and complexity growth of cyberattacks, intrusion and security analysts have to analyze a massive number of security alerts generated by heterogeneous security appliances. Analyzing this massive number of alerts is a resource-intensive task. Therefore, manual analysis for IDS alerts is not possible anymore. However, modern security threats and sophisticated attacks raise the need for alert analysis to filter, summarize, and detect relevant alerts. Alert correlation systems are proposed to analyze alerts from IDS and security appliances. In our research we proposed a novel approach to build alert correlation systems that can handle massive number of alerts generated by heterogeneous environment.

7.1 Work Summary

Using semantic analysis and ontological engineering to build alert correlation systems distinguishes our work from previous work in the area of alert correlation. We proposed a general approach to build method ontologies that support problem solving tasks and can effectively be used in building alert correlation systems. In addition, the proposed approach can be adapted or extended to other security-related analysis

and investigation tasks, such as network/computer forensics, vulnerability analysis, malware analysis, etc. There are two key advantages of using ontology in alert correlation or other security-related analysis tasks. The first advantage is interoperability between heterogeneous systems. The second advantage is representing the IDS alerts in a machine-readable format. This second advantage in particular made the semantic correlation and analysis of intrusion alerts possible.

Using semantic correlation and instance-based learning techniques such as KNN and rule induction, in addition to computational intelligence paradigms, we developed a new IDS alert verification methods that can eliminate false positive alerts. The experimental results demonstrate the promises of the proposed technique. The proposed technique can eliminate up to **98%** of false positive alerts and only miss less than **2%** of true positive alerts.

Controlling alert flooding in intrusion detection system is critical. We developed a novel technique for IDS alert fusion and summarization. The proposed technique relies on semantic similarity and the intrusion ontology as the key to cluster and abstract IDS alerts, and to generate a high level summarized representation of a massive number of alerts. The proposed technique also provides a mean for measuring information loss resulting from the aggregation of IDS alerts. Our experimental results shows that our technique can effectively reach up to **93%** alert reduction rate, with an information loss rate less than **11%**. The use of semantic correlation allows us to measure the quality of the aggregation process, which to our knowledge makes our technique the first alert aggregation technique that can express the quality of the aggregation process in a measurable manner.

A new technique for attack scenario and attack pattern reconstruction was proposed. The proposed technique uses semantic clustering and clique analysis to extract attack patterns and candidate attack scenarios. We also designed an attack causality

analysis technique that detects causality between attack steps/stages using attack impact. In addition, the proposed technique can detect missing attack steps and tolerate false negatives in IDS. The proposed technique showed promising results that encourage us to continue working on it. The main issue with our technique is that it is limited by the performance of the IDS. While we can argue that this limitation applies to all the attack reconstruction techniques in the literature, we believe we can extend the proposed technique to mitigate this limitation.

7.2 Future Work

The characteristics of the alerts generated by IDS and other security appliances are clear examples of the three V's (Volume, Velocity, Variety). The generated alerts are massive in volume, hundreds of thousands of alert are generated in few minutes, and the alerts come from many sources in different formats. Of course this means alert correlation is a clear case of big data analysis. In fact, recently a new term "Big Data Security Analysis" have been coined that refers to the analysis of security data using big data analysis techniques. Therefore, transforming our semantic correlation approach into the realm of big data analysis will be most likely the focus of our future work. Our alert verification, aggregation, and scenario reconstruction techniques need to be modified to work in big data environment, for example, using a Map-Reduce model to redesign and implement the algorithms proposed in our research.

Bridging the gap between alert correlation and other areas of cyberattacks analysis and response, such as network forensics, attack mitigation, and incident response is very important. To achieve that, we need a correlation framework that correlates alerts from different security appliances. Therefore, it is possible to extend our work to propose a security information and event management (SIEM) system using se-

mantic correlation that automates the aggregation and correlation of security events, provides threat analysis, and recommends mitigation/response based on previously investigated attacks and security incidents.

Finally, during our research and mainly in the experiment phase we reached several interesting observations. We found an interesting relation between alert correlation and visualization; both of them can benefit from each other. Visualization can largely improve the understanding of the data under investigation and the correlation can improve the visibility of the data. It will be interesting in the future to study the use of data visualization in alert correlation.

Bibliography

- [1] F. Abdoli and Mohsen Kahani. Using attacks ontology in distributed intrusion detection system. In *SCSS (1)*, pages 153–158, 2007.
- [2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. On finding lowest common ancestors in trees. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 253–265, New York, NY, USA, 1973. ACM.
- [3] Safaa O. Al-Mamory and Hong Li Zhang. Scenario discovery using abstracted correlation graph. In *Computational Intelligence and Security, 2007 International Conference on*, pages 702–706, Dec. 2007.
- [4] Safaa O. Al-Mamory and Hongli Zhang. Ids alerts correlation using grammar-based approach. *Journal in Computer Virology*, pages 271–282, 2009.
- [5] Vipin Kumar Aleksandar Lazarevic and Jaideep Srivastava. *Managing Cyber Threats*, volume 5 of *Massive Computing*, chapter Intrusion Detection: A Survey. Springer US, 2005. Part1.
- [6] Ahmad Almulhem and Issa Traore. Experience with engineering a network forensics system. In Cheeha Kim, editor, *Information Networking. Convergence in Broadband and Mobile Networking*, volume 3391 of *Lecture Notes in Computer Science*, pages 62–71. Springer Berlin Heidelberg, 2005.

- [7] R. Anbarestani, B. Akbari, and F. Fathi. An iterative alert correlation method for extracting network intrusion scenarios. In *Electrical Engineering (ICEE), 2012 20th Iranian Conference on*, pages 684–689, may 2012.
- [8] Arash Ghorbannia Delavar" "Asieh Mokarian, Ahmad Faraahi. False positives reduction techniques in intrusion detection systems-a review. *International Journal of Computer Science and Network Security*, 13(10):128–134, 2013.
- [9] S.J. Athenikos, Hyoil Han, and A.D. Brooks. Semantic analysis and classification of medical questions for a logic-based medical question-answering system. In *Bioinformatics and Biomeidcine Workshops, 2008. BIBMW 2008. IEEE International Conference on*, pages 111–112, Nov. 2008.
- [10] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Department of Computer Engineering, Chalmers University, March 2000.
- [11] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.
- [12] Mehdi Bateni, Ahmad Baraani, Ali Ghorbani, and Abbas Rezaei. An ais-inspired architecture for alert correlation. *International Journal of Innovative Computing, Information and Control*, 2013.
- [13] Kalle Burbeck and Simin Nadjm-tehrani. Adwice - anomaly detection with real-time incremental clustering. In *Proceedings of the 7th International Conference on Information Security and Cryptology, Seoul, Korea*. Springer Verlag, 2004.

- [14] Leandro R. de Castro and Jonathan Timmis. *Artificial Immune Systems: A New Computational Intelligence Paradigm*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
- [15] NASA's Johnson Space Center. C language integrated production system (clips), a tool for building expert systems. <http://cve.mitre.org>.
- [16] Yu-Chin Cheng, Chien-Hung Chen, Chung-Chih Chiang, Jun-Wei Wang, and Chi-Sung Lai. Generating attack scenarios with causal relationship. In *Granular Computing, 2007. GRC 2007. IEEE International Conference on*, pages 368–368, Nov. 2007.
- [17] Oliver Dain and Robert K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *In Proceedings of the 2001 ACM workshop on Data Mining for Security Applications*, pages 1–13, 2001.
- [18] Dipanjan Das and Andre' F. T. Martins. A Survey on Automatic Text Summarization, November 2007.
- [19] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pages 85–103, London, UK, 2001. Springer-Verlag.
- [20] Thierry Despeyroux. Practical semantic analysis of web sites and documents. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, 2004.
- [21] Yu-Xin Ding. Attack scenario detection based on expert system. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 6, pages 3283–3287, Aug. 2007.

- [22] Yu-Xin Ding, Hai-Sen Wang, and Qing-Wei Liu. Intrusion scenarios detection based on data mining. In *Machine Learning and Cybernetics, 2008 International Conference on*, volume 3, pages 1293–1297, July 2008.
- [23] A. Ebrahimi, A.H.Z. Navin, M.K. Mirnia, H. Bahrbeigi, and A.A.A. Ahrabi. Automatic attack scenario discovering based on a new alert correlation method. In *Systems Conference (SysCon), 2011 IEEE International*, pages 52–58, april 2011.
- [24] Gerhard Eschelbeck and Michael Krieger. Eliminating noise from intrusion detection systems. *Information Security Technical Report*, 8(4):26 – 33, 2003.
- [25] Guo Fan, Ye JiHua, and Yu Min. Design and implementation of a distributed ids alert aggregation model. In *Computer Science Education, 2009. ICCSE '09. 4th International Conference on*, pages 975–980, 2009.
- [26] Thomas R. Gruber. A translation approach to portable ontology specifications. *KNOWLEDGE ACQUISITION*, 5:199–220, 1993.
- [27] Michael Grüninger and Mark S. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of Workshop on Basic Ontological Issues in Knowledge Sharing held in conjunction with IJCAI-95*, 1995.
- [28] Néstor D. Duque Gustavo Isaza, Andrés Castillo. An intrusion detection and prevention model based on intelligent multi-agent systems, signatures and reaction rules ontologies. In *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*.
- [29] Simon Hansman and Ray Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31 – 43, 2005.

- [30] A. Hofmann and B. Sick. Online intrusion alert aggregation with generative data stream modeling. *Dependable and Secure Computing, IEEE Transactions on*, vol 8(num 2):282–294, 2011.
- [31] Hsien-Der Huang, Tsung-Yen Chuang, Yi-Lang Tsai, and Chang-Shing Lee. Ontology-based intelligent system for malware behavioral analysis. In *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, pages 1–6, july 2010.
- [32] Anita K. Jones and Robert S. Sielken. Computer system intrusion detection: A survey. Technical report, University of Virginia Computer Science Department, 1999.
- [33] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *The 8th ACM International Conference on Knowledge Discovery and Data Mining*, pages 366–375, 2002.
- [34] Kruegel and Christopher. *Intrusion Detection and Correlation: Challenges and Solutions*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.
- [35] Christopher Kruegel and William Robertson. Alert verification - determining the success of intrusion attempts. In *Proc. First Workshop the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004)*, pages 1–14, 2004.
- [36] Carl E. Landwehr, Alan R. Bull, John P. Mcdermott, and William S. Choi. A taxonomy of computer program security flaws, with examples. *ACM Comput. Surv.*, 26(3):211–254, September 1994.
- [37] Pat Langley and Herbert A. Simon. Applications of machine learning and rule induction. *Communications of the ACM*, 38:55–64, 1995.

- [38] Jie Lei and Zhi tang Li. Using network attack graph to predict the future attacks. In *Communications and Networking in China, 2007. CHINACOM '07. Second International Conference on*, pages 403–407, Aug. 2007.
- [39] Wan Li, Yan Zhu, and Shengfeng Tian. Intrusion alerts correlation model based on xswrl ontology. In *Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium on*, volume 1, pages 894–898, Dec. 2008.
- [40] Wang Li, Li Zhi-tang, Li Dong, and Lei Jie. Attack scenario construction with a new sequential mining technique. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, volume 1, pages 872–877, 30 2007-Aug. 1 2007.
- [41] Lincoln-Laboratory-MIT. Darpa intrusion detection evaluation. <http://www.ll.mit.edu/mission/communications/ist/CST/index.html>.
- [42] Zhijie Liu, Chongjun Wang, and Shifu Chen. Correlating multi-step attack and constructing attack scenarios based on attack pattern modeling. In *Information Security and Assurance, 2008. ISA 2008. International Conference on*, pages 214–219, April 2008.
- [43] Mariano F. Lopez, Asuncion G. Perez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium*, pages 33–40, Stanford, USA, March 1997.
- [44] Jie Ma, Zhi Tang Li, and Hong Wu Zhang. A fusion model for network threat identification and risk assessment. In *AICI '09: Proceedings of the 2009 International Conference on Artificial Intelligence and Computational Intelligence*, pages 314–318, Washington, DC, USA, 2009. IEEE Computer Society.

- [45] M. Mahboubian, N.I. Udzir, S. Subramaniam, and N.A.W.A. Hamid. An alert fusion model inspired by artificial immune system. In *Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on*, pages 317–322, june 2012.
- [46] John Mchugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, November 2000.
- [47] Russell Meyer. Challenges of managing an intrusion detection system (ids) in the enterprise. Technical report, SANS Institute InfoSec Reading Room, March 2008.
- [48] A.B. Mohamed, N.B. Idris, and B. Shanmugum. Alert correlation framework using a novel clustering approach. In *Computer Information Science (ICCIS), 2012 International Conference on*, volume 1, pages 403–408, june 2012.
- [49] A.B. Mohamed, N.B. Idris, and B. Shanmugum. Alert correlation using a novel clustering approach. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, pages 720–725, may 2012.
- [50] Thomas Moser, Heinz Roth, Szabolcs Rozsnyai, Richard Mordinyi, and Stefan Biffel. Semantic event correlation using ontologies. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II, OTM '09*, pages 1087–1094, Berlin, Heidelberg, 2009. Springer-Verlag.
- [51] B. Mukherjee, L.T. Heberlein, and K.N. Levitt. Network intrusion detection. *Network, IEEE*, 8(3):26–41, may-june 1994.

- [52] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 245–254, New York, NY, USA, 2002. ACM.
- [53] Peng Ning and Dingbang Xu. Hypothesizing and reasoning about attacks missed by intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 7(4):591–627, 2004.
- [54] Jean-Pierre Norguet, Benjamin Tshibusu-Kabeya, Gianluca Bontempi, and Esteban Zimnyi. A page-classification approach to web usage semantic analysis. *Engineering Letters*, 14(1):120–126, 2007.
- [55] D. Ourston, S. Matzner, W. Stump, and B. Hopkins. Applications of hidden markov models to detecting multi-stage network attacks. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 10 pp.–, Jan. 2003.
- [56] Vern Paxson. Bro an open-source, unix-based network intrusion detection system. <http://www.bro-ids.org/>.
- [57] Andrew Simmonds Peter, Peter S, and Louis Van Ekert. An ontology for network security attacks. In *Proceedings of the 2nd Asian Applied Computing Conference (AACC04), LNCS 3285*, pages 317–323. Springer-Verlag, 2004.
- [58] Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 102–124. Springer, 2004.
- [59] Victor Raskin, Christian F. Hempelmann, Katrina E. Triezenberg, and Sergei Nirenburg. Ontology in information security: a useful theoretical foundation

- and methodological tool. In *NSPW '01: Proceedings of the 2001 workshop on New security paradigms*, pages 53–59, New York, NY, USA, 2001. ACM.
- [60] A. Razzaq, H.F. Ahmed, A. Hur, and N. Haider. Ontology based application level intrusion detection system by using bayesian filter. In *Computer, Control and Communication, 2009. IC4 2009. 2nd International Conference on*, pages 1–6, feb. 2009.
- [61] S. Rekhis and N. Boudriga. A formal approach for the reconstruction of potential attack scenarios. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–6, April 2008.
- [62] Sang Keun Rhee, Jihye Lee, and Myon-Woong Park. Semantic relevance measure between resources based on a graph structure. In *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, pages 229–236, oct. 2008.
- [63] Tuukka Ruotsalo and Eero Hyvonen. A method for determining ontology-based semantic relevance. In *Proceedings of the 18th international conference on database and expert systems applications*, pages 3–7. Springer-Verlag, 2007.
- [64] S. Saad and I. Traore. A semantic analysis approach to manage ids alerts flooding. In *Information Assurance and Security (IAS), 2011 7th International Conference on*, pages 156–161, dec. 2011.
- [65] Sherif Saad and Issa Traore. A semantic-based approach to minimize ids alerts flooding. In *ASIACCS '12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*.

- [66] Sherif Saad and Issa Traore. Method ontology for intelligent network forensics analysis. In *Eight International Conference on Privacy, Security and Trust (PST 2010)*, pages 7–14, Ottawa, Canada, 8 2010.
- [67] Sherif Saad and Issa Traore. Ontology-based intelligent network-forensics investigation. In *19th International Conference on Software Engineering and Data Engineering (SEDE 2010)*, San Francisco, USA, 6 2010.
- [68] Sherif Saad and Issa Traore. Heterogeneous multi-sensor ids alerts aggregation using semantic analysis. *Journal of Information Assurance and Security*, 7(2):79–88, 2012.
- [69] Sherif Saad and Issa Traore. Extracting attack scenarios using intrusion semantics. In Joaquin Garcia-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Ali Miri, and Nadia Tawbi, editors, *Foundations and Practice of Security*, volume 7743 of *Lecture Notes in Computer Science*, pages 278–292. Springer Berlin Heidelberg, 2013.
- [70] Sherif Saad and Issa Traore. Semantic aware attack scenarios reconstruction. *Journal of Information Security and Applications*, 18(1):53 – 67, 2013. SE-TOP’2012 and FPS’2012 Special Issue.
- [71] Sherif Saad and Issa Traore. Context-aware false positives reduction in ids using a colonial section algorithm. *Computers and Security Journal*, 2015.
- [72] David Sánchez, Montserrat Batet, and David Isern. Ontology-based information content computation. *Know.-Based Syst.*, 24:297–303, March 2011.
- [73] Nuno Seco, Tony Veale, and Jer Hayes. An Intrinsic Information Content Metric for Semantic Similarity in WordNet. In *ECAI’2004, the 16th European Conference on Artificial Intelligence*, 2004.

- [74] Makoto Shimamura and Kenji Kono. Using attack information to reduce false positives in network ids. *Computers and Communications, IEEE Symposium on*, 0:386–393, 2006.
- [75] Greg Shipley. Anatomy of a network intrusion. *Netw. Comput.*, 10(21):124–128, 1999.
- [76] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security*, 31(3):357 – 374, 2012.
- [77] Sourcefire. Snort is an open source network intrusion prevention and detection system. <http://www.snort.org/>.
- [78] Ayman E. Taha, Ismail Abdel Ghaffar, Ayman M. Bahaa Eldin, and Hani M. K. Mahdi. Agent based correlation model for intrusion detection alerts. In *Intelligence and Security Informatics (ISI), 2010 IEEE International Conference on*, pages 89 –94, may 2010.
- [79] Zhi tang Li, Jie Lei, Li Wang, and Dong Li. A data mining approach to generating network attack graph for intrusion prediction. In *Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on*, volume 4, pages 307–311, Aug. 2007.
- [80] Gina C. Tjhai, Maria Papadaki, Steven Furnell, and Nathan L. Clarke. Investigating the problem of ids false alarms: An experimental study using snort. In *SEC*, pages 253–267, 2008.
- [81] Sheif Saad. Isaa Traore. and Marcelo Brocardo. Context-aware intrusion alerts verification approach. In *Information Assurance and Security (IAS), 2011 10th International Conference*, nov. 2014.

- [82] Jeffrey L Undercoffer, Anupam Joshi, Tim Finin, and John Pinkston. A Target-Centric Ontology for Intrusion Detection. In *The 18th International Joint Conference on Artificial Intelligence*, July 2003.
- [83] Mike Uschold and Michael Grüninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.
- [84] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pages 54–68, London, UK, 2001. Springer-Verlag.
- [85] F. Valeur, G. Vigna, C. Kruegel, and R.A. Kemmerer. Comprehensive approach to intrusion detection alert correlation. *Dependable and Secure Computing, IEEE Transactions on*, 1(3):146 – 169, jul. 2004.
- [86] Jouni Viinikka, Hervé Debar, Ludovic Mé, and Renaud Séguier. Time series modeling for ids alert management. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security, ASIACCS '06*, pages 102–113, New York, NY, USA, 2006. ACM.
- [87] Chao Wang, Jie Lu, and Guangquan Zhang. A semantic classification approach for online product reviews. In *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pages 276–279, Sept. 2005.
- [88] Sheng Wen, Yang Xiang, and Wanlei Zhou. A lightweight intrusion alert fusion system. In *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*, pages 695 –700, 2010.
- [89] Min Xiao and Debao Xiao. Alert verification based on attack classification in collaborative intrusion detection. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007*.

- Eighth ACIS International Conference on*, volume 2, pages 739–744, 30 2007-aug. 1 2007.
- [90] Shisong Xiao, Yugang Zhang, Xuejiao Liu, and Jingju Gao. Alert fusion based on cluster and correlation analysis. *Hybrid Information Technology, International Conference on*, pages 163–168, 2008.
- [91] Ming Xu and Wei Han. Distributed intrusion alert fusion based on multi keyword. In *ISDPE '07: Proceedings of the The First International Symposium on Data, Privacy, and E-Commerce*, pages 469–471, Washington, DC, USA, 2007. IEEE Computer Society.
- [92] Ming Xu, Ting Wu, and Jingfan Tang. An ids alert fusion approach based on happened before relation. pages 1–4, oct. 2008.
- [93] W. Yan, E. Hou, and N. Ansari. Frame-based attack representation and real-time first order logic automatic reasoning. In *Information Technology: Research and Education, 2005. ITRE 2005. 3rd International Conference on*, pages 225–229, June 2005.
- [94] Yugang Zhang, Shisong Xiao, Xin Zhuang, and Xi Peng. Using cluster and correlation to construct attack scenarios. In *Cyberworlds, 2008 International Conference on*, pages 471–476, Sept. 2008.
- [95] Tian Zhihong, Qin Baoshan, Ye Jianwei, and Zhang Hongli. Alertclu: A realtime alert aggregation and correlation system. In *Cyberworlds, 2008 International Conference on*, pages 778–781, 2008.
- [96] Xin Zhuang, Debao Xiao, Xuejiao Liu, and Yugang Zhang. Applying data fusion in collaborative alerts correlation. *Computer Science and Computational Technology, International Symposium on*, vol 2:124–127, 2008.