Using Root Cause Analysis to Handle Intrusion Detection Alarms

Dissertation

zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Universität Dortmund
am Fachbereich Informatik

von

Klaus Julisch

Dortmund

2003

Tag der mündlichen Prüfung:	15. Juli 2003
Dekan:	Prof. Dr. Bernhard Steffen
Gutachter:	Prof. Dr. Joachim Biskup
	Prof. Dr. Heiko Krumm

Abstract

Using Root Cause Analysis to Handle Intrusion Detection Alarms

Klaus Julisch
IBM Zurich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon, Switzerland
e-mail: kju@zurich.ibm.com

In response to attacks against enterprise networks, administrators are increasingly deploying intrusion detection systems. These systems monitor hosts, networks, and other resources for signs of security violations. Unfortunately, the use of intrusion detection has given rise to another difficult problem, namely the handling of a generally large number of mostly false alarms. This dissertation presents a novel paradigm for handling intrusion detection alarms more efficiently.

Central to this paradigm is the notion that each alarm occurs for a reason, which is referred to as the alarm's *root causes*. This dissertation observes that a few dozens of root causes generally account for over 90% of the alarms in an alarm log. Moreover, these root causes are generally persistent, i.e. they keep triggering alarms until someone removes them. Based on these observations, we propose a new two-step paradigm for alarm handling: Step one identifies root causes that account for large numbers of alarms, and step two removes these root causes and thereby reduces the future alarm load. Alternatively, alarms originating from benign root causes can be filtered out. To support the discovery of root causes, we propose a novel data mining technique, called *alarm clustering*.

To lay the foundation for alarm clustering, we show that many root causes manifest themselves in alarm groups that have certain structural properties. We formalize these structural properties and propose alarm clustering as a method for extracting alarm groups that have these properties. Such alarm groups are generally indicative of root causes. We therefore present them to a human expert who is responsible for identifying the underlying root causes. Once identified, the root causes can be removed (or false positives can be filtered out) so as to reduce the future alarm load. We experimentally validate the proposed two-step alarm handling paradigm with alarms from a variety of different operational environments. These experiments show that alarm clustering makes the identification of root causes very efficient. Moreover, the experiments demonstrate that by judiciously responding to root causes one can reduce the future alarm load by 70%, on the average.

Acknowledgments

This thesis work began under the guidance of Prof. Marc Dacier, who at the time, was my manager at the IBM Zurich Research Laboratory. Prof. Dacier always gave me the feeling that my research was important and exciting, and he pushed me hard to get results early on. Moreover, I benefited greatly from his long experience in the field. For all that, I thank you, Marc.

As companies do not award academic titles, I had to register as a PhD student at a university. When I presented my fledgling research project to Prof. Joachim Biskup, he saw value in it, and agreed to supervise me. Prof. Biskup has a natural talent for asking all the difficult questions, and while he pushed me hard to make progress with these questions, he also showed considerable understanding for my somewhat special situation as an external PhD student. He was always prompt in answering my questions, and generously allocated time for discussions with me. Thanks to that, I could finish my thesis work before my contract at IBM expired. Thank you, Joachim, is the least I can say.

I also thank my colleagues at the IBM Zurich Research Laboratory, all of whom have contributed to this thesis in one way or another. In particular, I thank Andreas Wespi, my current manager, for giving me a lot a freedom to work on my thesis. Dominique Alessandri and James Riordan have been wonderful friends and a great help in administering the Linux server, on which I ran my experiments. I am very grateful to Birgit Baum-Waidner for her excellent comments on an earlier draft of this document. Finally, I thank Larry Oliver and Alex Wood of IBM Managed Security Services Delivery (formerly Emergency Response Team). They supported me with their expertise, and shared real-world data with me, so that I could validate my thesis work.

Last but not least, I acknowledge the support from the European IST Project MAFTIA (IST-1999-11583), which is partially funded by the European Commission and the Swiss Department for Education and Science.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Over the past 10 years, the number and severity of network-based computer attacks have significantly increased [6]. As a consequence, classic computer security technologies such as authentication and cryptography have gained in importance. Simultaneously, intrusion detection has emerged as a new and potent approach to protect computer systems [14, 47]. In this approach, so-called *Intrusion Detection Systems (IDSs)* are used to monitor computer systems for signs of security violations. Having detected such signs, IDSs trigger alarms to report them. These alarms are presented to a human operator who evaluates them and initiates an adequate response. Examples of possible responses include law suits, firewall reconfigurations, and the fixing of discovered vulnerabilities.

Evaluating intrusion detection alarms and conceiving an appropriate response was found to be a challenging task. In fact, practitioners [29, 141] as well as researchers [13, 22, 37, 111] have observed that IDSs can easily trigger thousands of alarms per day, up to 99% of which are *false positives* (i.e. alarms that were mistakenly triggered by benign events). This flood of mostly false alarms makes it very difficult to identify the hidden *true positives* (i.e. those alarms that correctly flag attacks). For example, the manual investigation of alarms has been found to be labor-intensive and error-prone [29, 44, 141]. Tools to automate alarm investigation are being developed [44, 48, 199], but there is currently no silver-bullet solution to this problem.

This thesis presents a new semi-automatic approach for handling intrusion detection alarms more efficiently. Central to this approach is the notion of alarm root causes. Intuitively, the *root cause* of an alarm is the reason for which it occurred. For example, consider a machine whose broken TCP/IP stack generates fragmented IP traffic. This traffic will trigger "Fragmented IP" alarms on many IDSs. Here, the broken TCP/IP stack is the root cause of the respective alarms. This thesis observes that for any given IDS, the majority of alarms can be attributed to one out of a small number of root causes. Moreover, many of these root causes are persistent and do not disappear unless someone removes them. It is therefore argued that intrusion detection alarms should be handled by identifying and removing the most predominant and persistent root causes. In that way, the future alarm load is reduced, and the human operator is freed to focus on the remaining alarms. Applied to the above example, this means to identify and fix the broken TCP/IP stack, which also eliminates the associated "Fragmented IP" alarms. To make this new alarm handling paradigm practical, we introduce a new data mining technique that — when applied to alarm logs — supports the discovery of root causes.

1.2 Thesis Statement and Contributions

This dissertation describes the work done to validate the following three-part thesis statement:

> *(1)* A small number of root causes is generally responsible for the bulk of alarms triggered by an IDS. *(2)* Root causes can be discovered efficiently by performing data mining on alarm logs. *(3)* Knowing the root causes of alarms, one can — in most cases — safely and significantly reduce the future alarm load by removing them or by filtering out the alarms that originate from benign root causes.

Part one of this thesis statement is a general observation about the nature of intrusion detection alarms, namely that only a few root causes account for the bulk of alarms. Part two states that root causes can be discovered *efficiently*, and part three says that knowledge of root causes is relevant because it enables countermeasures that are *effective* in reducing the future alarm load. For example, a possible countermeasure is to fix root causes such as broken protocol stacks, compromised machines, or configuration faults (which some IDSs confuse with attacks). Alternatively, one can filter out alarms that are known to result from benign (i.e. not security relevant) root causes. Finally, reducing the future alarm load based on one's understanding of alarm root causes is claimed to be *safe*, meaning that it incurs a small risk of discarding true positives.

Note that the thesis statement implies a new paradigm for alarm handling. According to this paradigm, data mining should be used to discover root causes that account for large numbers of alarms. Subsequently, removing these root causes or judiciously filtering out their associated alarms will safely and significantly reduce the number of alarms that are triggered in the future. The thesis statement guarantees that this approach is efficient, effective, and safe.

The proof of the thesis statement is constructive and experimental. A substantial fraction of this dissertation is concerned with deriving a data mining technique that supports the discovery of root causes. Subsequently, this data mining technique is used in experiments with real-world intrusion detection alarms to validate the thesis statement. As Section 1.4 will point out, we are not in possession of representative data collections from all kinds of IDSs. Therefore the scope of our experimental validation is restricted by the available data.

The main ideas underlying this dissertation have been published in several articles [110, 111, 112, 113, 114] and two patents [42, 43]. The novel contributions can be summarized as follows:

- The rule that a few root causes generally account for the majority of alarms is established.
- We formalize the link between data mining and root cause discovery, investigate the suitability of existing data mining techniques for root cause discovery, and develop a new, particularly suitable data mining technique.
- We use the aforementioned data mining technique to show that root causes can be discovered in an efficient manner by analyzing alarm logs.
- In extensive experiments with real-world intrusion detection alarms, it is shown that intrusion detection alarms can be handled safely and efficiently by identifying and responding to their root causes.
- This dissertation offers the first high-level characterization of intrusion detection alarms and shows how it applies to the detection of attacks (cf. Proposition 3.1 and Section 3.5.2).
- To the field of data mining, we contribute a detailed case study for the successful application of its techniques (including episode rules, tests of cluster tendency, cluster analysis, and cluster validation).

1.3 Overview

Intrusion detection systems trigger alarms to report attacks. This dissertation models alarms as tuples over the Cartesian product $\text{dom}(A_1) \times \ldots \times \text{dom}(A_n)$, where $\{A_1, \ldots, A_n\}$ is the set of alarm attributes and $\text{dom}(A_i)$ is the domain (i.e. the range of possible values) of attribute A_i. The *alarm attributes* (*attributes* for short) capture intrinsic alarm properties, such as the source IP address of an alarm, its destination IP address, its alarm type (which encodes the observed attack), and its time-stamp. Finally, $\mathbf{a}[A_i]$ denotes the value that attribute A_i assumes in alarm $\mathbf{a}$. For example, Table 1.1(a) shows a sample alarm that reports a "Fragmented IP" attack being launched on Nov. 1st at 11:33:17 from the source 10.173.2.3 : 80 against the target 10.7.121.9 : 12318. Moreover, if $\mathbf{a}$ denotes the alarm of Table 1.1(a), then $\mathbf{a}[\text{Time-stamp}] = \text{Nov. 1st, 11:33:17}$ holds.

At the core of this dissertation is a data mining technique called *alarm clustering*. The motivation for alarm clustering stems from the observation that the alarms of a given root cause are generally "similar" in a sense defined below. Alarm clustering reverses this implication and groups similar alarms together, assuming that these alarms also share the same root cause. For each alarm cluster (i.e. for each group of similar alarms), a single generalized alarm is derived. Intuitively, a generalized alarm is a succinct and human-understandable pattern that an alarm must match in order to belong to the respective cluster. In practice, knowledge of generalized alarms vastly simplifies the discovery of root causes. We therefore use alarm clustering to identify the root causes in real-world alarm logs. In doing so, we empirically validate the thesis statement. The following example is instrumental for a better understanding of alarm clustering.

Example 1 *Let us reconsider the root cause of the broken TCP/IP stack, which was first introduced in Section 1.1. This time, we assume that the broken stack belongs to a popular Web server that is primarily used on workdays. Obviously, the resulting "Fragmented IP" alarms have the same source IP address (namely the IP address of the Web server, which we assume to be 10.173.2.3) and the same source port (namely 80). The target of the alarms are non-privileged ports of various Web clients. Furthermore, given that the Web server is mostly used on workdays, one will observe that the bulk of alarms occurs on workdays. Finally, note that "Fragmented IP" alarms are triggered each time that the Web server responds to a client request. Given our assumption that the Web server is popular and therefore heavily used, it follows that we are flooded by a large number of "Fragmented IP" alarms.*

The alarm clustering method developed in this dissertation groups the "Fragmented IP" alarms together and reports them by saying that there are many alarms that are subsumed by the generalized alarm of "source 10.173.2.3 : 80 launching 'Fragmented IP' attacks on workdays against non-privileged ports of Web clients". Clearly, a generalized alarm like this facilitates the identification of root causes, even though complete automation is not achieved. □

More formally, generalized alarms — just like ordinary ones — are tuples that indicate for each attribute the corresponding attribute value. Thus, the generalized alarm of the above example actually looks as shown in Table 1.1(b). Note that

Table 1.1: Illustration of alarms and generalized alarms.

a) A sample alarm.

Attribute	**Value**
Source IP	10.173.2.3
Source port	80
Destination IP	10.7.121.9
Destination port	12318
Alarm type	Fragmented IP
Time-stamp	Nov. 1st, 11:33:17

b) A sample generalized alarm.

Attribute	**Value**
Source IP	10.173.2.3
Source port	80
Destination IP	Web clients
Destination port	Non-privileged
Alarm type	Fragmented IP
Time-stamp	Workdays

different from ordinary alarms, generalized alarms can have generalized attribute values. A *generalized attribute value* is an identifier that represents a set of elementary values. For example, the generalized attribute value "Workdays" of Table 1.1(b) represents the set of elementary time-stamps that fall on workdays. Similarly, the generalized attribute value "Non-privileged" represents the set $\{1025, \ldots, 65535\}$ of non-privileged ports. Generalized attribute values like these are not allowed in ordinary alarms.

There are two important observations to be made about the above example: First, the broken TCP/IP stack causes alarms that are *similar* in the sense that they are subsumed by the same generalized alarm. Second, the broken TCP/IP stack causes *many* alarms to be triggered. In other words, the broken TCP/IP stack is a root cause that manifests itself in a *large group of similar alarms*. This observation is central to the Chapters 3 and 4. More precisely, the remainder of this dissertation (excluding the related work in Chapter 2, and the conclusion in Chapter 7) is organized as follows:

Chapter 3 explains that data mining should ideally form alarm clusters whose constituent alarms are guaranteed to have the same root cause. Unfortunately, this goal is unattainable. Towards a weaker but feasible goal, we generalize the example of the broken TCP/IP stack and show that there is an important family of root causes that manifest themselves in *large groups of similar alarms*. Hence, we postulate that a data mining technique that identifies such large groups of similar alarms is likely to actually group alarms that have the same root cause. Therefore, cluster analysis, which per definition groups similar object, emerges as a promising data mining technique for root cause analysis.

Chapter 4 surveys the field of cluster analysis and shows that none of the popular clustering methods is really suitable for root cause analysis. This raises the need for a new clustering method, which is subsequently derived by modifying the method of attribute-oriented induction. In a nutshell, the new clustering method seeks to identify alarm clusters that maximize intra-cluster similarity, while having a user-defined minimum size. Similarity is measured via generalization hierarchies, which, for example, might state that the IP address 10.173.2.3 is a "Web server", is a "Server", is a "Machine", etc. . Then, the alarms of an alarm cluster are all the more similar the closer their attributes are related by way of these generalization hierarchies.

Chapter 5 attempts to objectively measure the quality of alarm clusters. Objective measures of cluster quality are desirable because they reduce the risk of a human user misinterpreting and consenting to meaningless alarm clusters. Cluster quality is studied in the field of *cluster validation*. Chapter 5 reviews this field, discusses its rather substantial limitations, and shows how the methods of cluster validation can nonetheless be used to increase our confidence in the results of the alarm clustering method.

Chapter 6 uses the alarm clustering method of Chapter 4 in experiments with real-world intrusion detection alarms. In doing so, we empirically validate the thesis statement given in the previous section.

1.4 Datasets Used in the Expriments

A preliminary remark on intrusion detection terminology is in order: IDSs are commonly classified into misuse detection systems and anomaly detection systems [14]. *Misuse detection systems* (a.k.a. signature-based systems) such as STAT [102] use knowledge accumulated about attacks to detect instances of these attacks. *Anomaly detection systems* (e.g. IDES [109]) use a reference model of normal behavior and flag deviations from this model as anomalous and potentially intrusive. Another dichotomy splits IDSs according to their audit sources. Specifically, *host-based IDSs* analyze host-bound audit sources such as operating system audit trails, system logs, or application logs, whereas *network-based IDSs* analyze network packets that are captured from a network.

The experiments in this dissertation use alarms from network-based, commercial misuse detection systems that were deployed in operational (i.e. "real-world") environments. We consider it a strength of our validation that it uses alarms from real-world environments rather than from simulated or laboratory environments, which can have significant limitations [144]. We are not in possession of data collections from host-based or anomaly detection systems and therefore cannot offer experiments with these IDS types.

Table 1.2 introduces the sixteen IDSs that we use in the experiments of this dissertation. Our selection criteria was to offer a representative mix of IDSs from

Table 1.2: Overview of IDSs used in experiments.

IDS	Type	Location	Min	Max	Avg
1	A	Intranet	7643	67593	39396
2	A	Intranet	28585	1946200	270907
3	A	DMZ	11545	777713	310672
4	A	DMZ	21445	1302832	358735
5	A	DMZ	2647	115585	22144
6	A	Extranet	82328	719677	196079
7	A	Internet	4006	43773	20178
8	A	Internet	10762	266845	62289
9	A	Internet	91861	257138	152904
10	B	Intranet	18494	228619	90829
11	B	Intranet	28768	977040	292294
12	B	DMZ	2301	289040	61041
13	B	DMZ	3078	201056	91260
14	B	Internet	14781	1453892	174734
15	B	Internet	248145	1279507	668154
16	B	Internet	7563	634662	168299

different vendors in different operational environments. The sixteen IDSs are deployed at eleven different Fortune 500 companies, and no two IDSs are deployed at the same geographic site. All IDSs are configured and operated by professional IDS analysts. The "IDS" column contains a numerical identifier that we use throughout this thesis to reference the IDSs. The "Type" column indicates the IDS type, namely "A" or "B", both of which are leading commercial IDSs. To avoid unintended commercial implications, we do not reveal the product names or vendors of "A" and "B". For each IDS, we employ all alarms that were triggered during the year 2001. The minimum, maximum, and average number of alarms per month are listed for each IDS in the "Min", "Max", and "Avg" columns, respectively. Finally, the "Location" column indicates where the IDSs are deployed:

Intranet: Denotes an IDS on an internal corporate network without Internet access.

DMZ: Denotes an IDS on a perimeter network that is protected by a firewall, but offers services to the Internet.

Extranet: Denotes an IDS on a network that is shared between multiple cooperating companies, but is not accessible from the Internet.

Internet: Denotes an IDS that is deployed before the external firewall on a direct link to the Internet.

Despite continuing standardization efforts [60], there is no widely used format for intrusion detection alarms. In particular, the alarms of different IDSs do not necessarily have the same alarm attributes. However, the alarms of virtually all network-based IDSs contain the following key attributes, which are the only ones that will be used in the experiments:

1. The *source IP address* (*source IP* for short) which identifies the machine that launced the attack.
2. The *source port* from which the attack originated.
3. The *destination IP address* (*destination IP* for short) of the target machine.
4. The *destination port* at which the attack was targeted.
5. The *time-stamp* indicating the date and time of the attack.
6. The *alarm type*, which describes the nature of the attack. The alarm type can be a mnemonic name such as “Fragmented IP” or a number that maps to a name like this, or to a more verbose description of the attack.
7. The *diagnostic context* (*context* for short), which stores the supposedly malicious network packet. For example, the context frequently contains supposedly malicious URLs, or command sequences that seem to constitute FTP or Telnet exploits.

Attributes are commonly classified according to their type. The *type* of an attribute characterizes the structure of its domain. For example, *numerical* attributes have the real numbers $\mathbb{R}$ as their domain. Attribute types play an important role in data mining because they influence the applicability of data mining techniques. Intrusion detection alarms contain attributes of many different types:

Categorical attributes: The domain of categorical attributes is discrete and unordered. Examples of categorical attributes include IP addresses and port numbers.

Numerical attributes: Examples of numerical attributes include counters (e.g. for the number of SYN packets in a SYN flooding alarm), and size attributes (e.g. for the packet size in “Large ICMP Traffic” alarms [30]).

Time attributes: All alarms are time-stamped. Note that time should not be treated as a numerical attribute because this would mean to ignore its unique semantics (including notions such as periodicity, workdays versus weekends, etc.).

Free-text attributes: Free-text attributes assume arbitrary and unforeseeable text values. The context, for example, is a free-text attribute.

Alarms are stored in *alarm logs*. The experiments in this dissertation are such that they take an alarm log as input and return an analysis of this alarm log as output. In other words, the alarm log is our basic unit of analysis. It is therefore necessary to split the alarms of Table 1.2 into alarm logs. This can be done in many different ways. We have chosen to split the alarms along IDS and month boundaries, so that we obtain a total of $(16\text{ IDSs}) \times (12\text{ months}) = 192$ different alarm logs. A corollary from this splitting is that alarms from different IDSs or from different months are never analyzed together because they do not fall into the same alarm log. Two factors were decisive for this splitting: First, analyzing alarm logs that span a prolonged time period such as a month allows us to discover long-term alarm patterns, such as weekly alarm patterns. Second, analyzing alarm logs that mix alarms from different IDSs seems pointless because the IDSs of Table 1.2 are deployed in completely independent environments. Therefore, any correlations between the alarms of these IDSs are mostly coincidental.

The notation and conventions used in this dissertation can be summarized as follows:

- We model alarms as tuples over the n-dimensional attribute space $\mathrm{dom}(A_1) \times \ldots \times \mathrm{dom}(A_n)$, where $\mathrm{dom}(A_i)$ is the domain of attribute A_i. The value that attribute A_i assumes in alarm $\mathbf{a}$ is denoted by $\mathbf{a}[A_i]$.
- The alarms in our experiments have the following attributes: The source IP address, the source port, the destination IP address, the destination port, the time-stamp, the alarm type, and the diagnostic context.
- Our experiments are such that they take the alarms triggered by one IDS during one month as input and return an analysis of these alarms as output.
- This text uses the following typeface conventions: Italic small letters (e.g. $x_1, y_2, z_3, \ldots$) denote atomic values such as numbers or IP addresses. Bold small letters (e.g. $\mathbf{a}, \hat{\mathbf{a}}, \mathbf{b}, \ldots$) denote tuples. Italic capital letters (e.g. $A_1, \mathit{Src{-}IP}, L, C, \ldots$) stand for attribute names, sets, and partitions (i.e. sets of sets).
- In anticipation of Chapter 3, we remark that alarm logs will be modeled as multi-sets (a.k.a. bags or collections). We use pointed brackets for multi-sets (e.g. $L_1 = <\mathbf{a}, \mathbf{a}, \mathbf{b}>$, $L_2 = <\mathbf{b}, \mathbf{c}>$) and re-define the set operators so that they take multiple occurrences of the same element into account. Hence, for example, $|L_1| = 3$, $|L_2| = 2$, $L_1 \cup L_2 = <\mathbf{a}, \mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}>$, $L_1 \nsubseteq L_2$, $<\mathbf{a}, \mathbf{a}> \subseteq L_1$, etc. .

Chapter 2

Related Work

This dissertation was motivated by the fact that today's IDSs tend to trigger an abundance of mostly false alarms. It is therefore natural to ask for the reasons of this alarm flood. Section 2.1 addresses this question, and surveys research towards "better" IDSs, which trigger less false positives. Section 2.2 introduces root cause analysis and shows how it has been used in network management. Section 2.3 surveys applications of data mining to intrusion detection. In particular, it describes earlier work that used data mining to render alarm investigation more efficient. Section 2.4 concludes this chapter with a discussion of alarm correlation.

2.1 On the Difficulty of Intrusion Detection

This section puts the thesis work into context. Specifically, Section 2.1.1 explains why IDSs trigger so many mostly false alarms, and Section 2.1.2 summarizes work into "better" IDSs, which trigger less false positives.

2.1.1 Origins of the Alarm Flood

This section only considers the false alarm problem for misuse detection systems. For these systems, the abundance of alarms in general, and of false positives in particular, can be attributed to three main factors:

Underspecified signatures: Underspecified signatures check conditions that are necessary, but not sufficient for attacks. As a consequence, they also trigger on benign events, which causes false positives. For example, instead of complex regular expressions that can reliably detect many attacks, it is not uncommon to use simple string-matching signatures. There are four reasons for this practice: First, harsh real-time requirements generally preclude the use of precise signatures, which are more time-consuming to match against the audit data [103, 176]. Second, because of their higher generality, it is attractive to use underspecified signatures that also cover *variations* of attacks [37, 48]. Third, audit sources frequently lack information useful for misuse detection [175, 176]. That can make underspecified signatures inevitable. Fourth, writing intrusion detection signatures is inherently difficult [122, 129, 157, 163], which favors the creation of buggy and possibly underspecified signatures.

Intent-guessing signatures: Intent-guessing signatures trigger on events that might or might not be attacks. For example, signatures that trigger on failed user logins, DNS zone transfers, overlapping IP fragments, or set URGENT bits are intent-guessing because they assume that these activities are malicious. It has been shown that this assumption is frequently false [19, 168]. Note that intent-guessing signatures are *not* underspecified as they reliably detect the events they claim to detect. However, these events are not necessarily attacks.

Lack of abstraction: Today's IDSs tend to trigger multiple low-level alarms to report a single conceptual-level phenomenon. For example, a single run of the nmap scanning tool [72] triggers hundreds of alarms, namely one alarm for each probe. Similarly, a network with a small maximum transfer unit (MTU) [196] systematically fragments IP packets. Nevertheless, most IDSs trigger a separate alarm for each fragmented packet. Clearly, this lack of abstraction aggravates the alarm flood.

Axelsson has observed that the use of underspecified or intent-guessing signatures easily leads to an over-proportionally high number of false positives [13]. Just observe that these signatures are prone to trigger on non-intrusive events. Unfortunately, non-intrusive events are so much more frequent than intrusive events that even their occasional misclassification can easily lead to an unacceptably high false alarm rate. This can be shown by using Bayes' Theorem to rewrite the conditional probability $P(\neg I|A)$ that an alarm A reports a non-intrusive event $\neg I$ (i.e. that A is a false positive):

$$P(\neg I|A) = \frac{P(\neg I) \cdot P(A|\neg I)}{P(\neg I) \cdot P(A|\neg I) + P(I) \cdot P(A|I)} \tag{2.1}$$

Assuming that the prior probability $P(\neg I)$ of non-intrusive events is much higher than the prior probability $P(I)$ of intrusive events (say, $P(\neg I) = 100 \cdot$

$P(I)$), we obtain $P(\neg I|A) \geq P(A|\neg I)/(P(A|\neg I)+0.01)$. Thus, a signature that is inclined to trigger on non-intrusive events (say, $P(A|\neg I) = 0.1$) will result in an unacceptably high rate of false positives (namely $P(\neg I|A) \geq 0.1/0.11 > 0.9$, for the above numerical values). Axelsson argues that a 90% probability of an alarm being a false one is completely unacceptable as it schools the human operator to ignore the alarms altogether [13].

2.1.2 Towards Better IDSs

The intuitively most appealing way of dealing with false positives probably is to build "better" IDSs. This is no easy endeavor because many of the issues mentioned in the previous section are intrinsically difficult to address. Nevertheless, there is a small number of research projects that have pursued this avenue. Interestingly, most of the resulting IDSs share two properties: First, they have public signatures that can be tuned to a given environment, and second they are special-purpose. *Special-purpose IDSs* are tailored towards detecting one class of attacks (e.g. Web server attacks) and they monitor audit sources that are particularly suitable for this task. By way of illustration, here are three examples of IDSs that are less prone to false positives:

Embedded detectors: Zamboni [204] defines embedded detectors as host-based IDSs that are integrated into the source code of an application or the operating system. Thus, embedded detectors are a form of source code instrumentation. One of their main advantage is their ability to access *any* information that they need to do their job. Moreover, embedded detectors are executed on demand, which is economical, and frees resources to be used otherwise, e.g. for more accurate signatures. Both advantages are expected to translate into fewer false positives [8, 204], but a rigorous proof is still amiss.

Web IDS: Almgren et al. describe a signature-based IDS for detecting Web server attacks in real-time [7]. The IDS is host-based and uses Web server logs as audit source. The attack signatures are a variant of regular expressions, and can easily be tuned to a particular environment. This customizability was found to be helpful for reducing the number of false positives.

Special-purpose NIDS: Sekar et al. present a network-based IDS that exclusively focuses on low-level network attacks, such as reconnaissance scans and denial-of-service attacks [183]. The presented system differs from most other network-based IDSs in that it refrains from any attempt to detect application-level attacks such as attacks against Web servers. Owing to its clear focus, the IDS is reported to have a very low false alarm rate.

Note that special-purpose IDSs such as the ones above have to be complemented by additional IDSs to achieve comprehensive attack coverage. This creates the need to deploy and manage a heterogeneous collection of complementary IDSs. Even though this is mainly an operational problem, one should not underestimate its practical difficulty and relevance. Moreover, none of the above IDSs addresses the difficulty of writing correct attack signatures. This is particularly evident for embedded detectors, which are integrated into the source code of the monitored program. Therefore, embedded detectors are more difficult to implement, maintain, and port [204].

2.2 Root Cause Analysis

This section briefly surveys the field of root cause analysis. Specifically, Subsection 2.2.1 introduces root cause analysis basics, and Subsection 2.2.2 shows how root cause analysis has been used in network management to identify faulty network equipment. The application of root cause analysis to network management is particularly relevant to this thesis because network management also faces the problem of an overwhelmingly large number of alarms.

2.2.1 Introduction to Root Cause Analysis

Root Cause Analysis (RCA) has historically been used to identify the most basic factors that contributed to an incident [128, 138, 167]. For example, after a reactor accident, the Department of Energy would commonly use root cause analysis to pinpoint why the accident had happened, so that similar accidents could be prevented in the future. Because prevention of similar accidents is central to root cause analysis, it has been required that root causes must be factors that one has control over. Hence, the following definition by Paradies and Busch [167]:

Definition 2.1 *A* **root cause** *is the most* basic *cause that can* reasonably *be identified and that management has control to* fix. **Root cause analysis** *is the task of identifying root causes.* □

The three key words in the definition of root causes are *basic*, *reasonably*, and *fix*: Root causes must be so *basic* (or specific) that one can *fix* them. On the other hand side, given that fixing them is the whole point, it is not *reasonable* to further split root causes into more basic causes. Consequently, these more basic causes are *not* root causes, and root causes lie at the “highest” level where fixing is possible. Finally, note that a single incident can have multiple root causes.

Several methodologies and supporting tools have been suggested for root cause analysis [128, 138]. Common to these methodologies is the notion that events propagate by causing other events. For example, a "poor-manufacturing-procedure event" *causes* a "buggy-fuel-gauge event", which *causes* precipitous "fuel-low-warning events", which *cause* an "emergency-airplane-landing event". The majority of root cause analysis methodologies provide systematic and structured frameworks for "walking back" such causality graphs to their roots. In addition, there are several computer tools to support these methodologies. See [128, 138] for details and examples.

2.2.2 Root Cause Analysis in Network Management

In network management [24, 98, 106, 107, 134, 164, 165, 202], alarms indicate problems in a network's operation, such as hardware or software failures, performance degradations, or misconfigurations. As network components are highly inter-dependent, a problem in one component propagates to all transitively dependent components. As a consequence, a problem affecting any single component can impair many other components, most of which report their impairment by means of alarms. The goal of network management is to evaluate these alarms and to pinpoint the original problem. This problem is called the root cause, and fixing it clearly eliminates the associated impairments and alarms. Finally, the task of identifying root causes is called root cause analysis [107, 202]. These definitions are consistent with the ones of the previous section.

The example of Figure 2.1 illustrates the challenges of root cause analysis. Figure 2.1(a) shows a network consisting of three routers (namely $R1$, $R2$, $R3$), three LANs (namely A, B, C), and a database server *DB*. The database clients are assumed to be distributed across all three LANs. Figure 2.1(b) shows how a problem in the memory module of router $R1$ propagates through the network. Specifically, suppose that in router $R1$, the memory cell storing the cost of link $R1 \rightarrow R3$ fails, and erroneously indicates an extremely high value. No alarm is issued for this event, but the routing protocol reacts by routing all future $A — C$ traffic over router $R2$. As a consequence, router $R2$ becomes congested, which results in alarms. Moreover, the connectivity of LAN C degrades and database transactions by remote database clients last longer. This increases the number of transaction aborts that are issued to release database locks that were held for too long. The increased number of transaction aborts is reported by additional alarms. Note that all alarms in this example report symptoms of the root cause (the failed memory cell) rather than the root cause itself. Moreover, router $R2$ and the database clients are likely to raise multiple alarms to report their impairment, which can unleash an alarm flood. This tendency to generate too many alarms

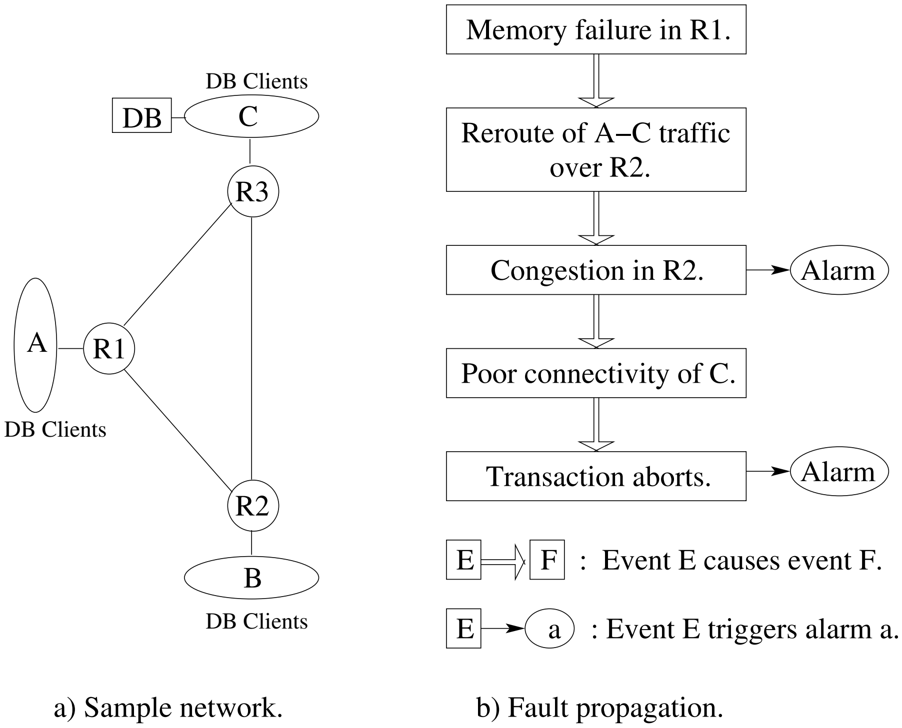


Figure 2.1: Fault propagation and alarm generation in networks.

that are only remotely related to the actual root cause makes root cause analysis a challenging task.

In summary, root cause analysis in network management aspires to identify the starting point(s) of the fault propagation chain (cf. Figure 2.1(b)). This is an instance of the general problem of abductive inference [160, 165, 169, 170]. *Abductive inference* is the process of reasoning from effects to causes. Many network management systems do abductive inference in two steps: First, they model the cause-effect propagation in networks, and then, they heuristically search this model for plausible root causes that explain the observed alarms [24, 98]. Other systems require the user to encode his or her knowledge about root cause analysis in expert system rules [106, 107, 164]. Thereby, the problem of abductive inference is off-loaded upon the user. Yet other systems implement root cause analysis by means of case-based reasoning [134] or codebooks [202].

Even though much research has investigated root cause analysis in network management, this research is not directly applicable to intrusion detection. Specifically, network management systems diagnose root causes by "walking back" the fault propagation chain. In intrusion detection, this paradigm is difficult to apply because the notions of causality and propagation are unclear. Moreover, network management systems can only diagnose *known* root causes. However, this dissertation has introduced the root cause concept to intrusion detection, and at the outset, there were no known root causes. Therefore, we decided to develop a tool that supported the discovery of new and unexpected root causes, rather than a tool that recognized known root causes.

2.3 Data Mining

Advances in computers and data storage technology have made it possible to collect huge datasets, containing gigabytes or even terabytes of data. This wealth of data contains potentially valuable information. The trick is to extract this (hidden) valuable information, so that the data owners can capitalize on it. Data mining is a relatively new discipline that seeks to do just that. More precisely, and accepting that slightly different definitions have been proposed (e.g. [63, 90]), we adopt the following working definition [91]:

Definition 2.2 Data mining *is the analysis of (often large) observational datasets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.* □

The relationships and summaries derived through a data mining exercise are often referred to as *models* or *patterns*. Examples include linear equations, clusters, tree structures, if-then rules, and periodic patterns.

The above definition refers to "observational data", as opposed to "experimental data". Data mining typically deals with data that has been collected for some purpose other than data mining. Thus, the objective of the data mining task could not influence the data collection strategy. This contrasts with statistics, where it is quite common to design experiments so that they yield the data needed to answer a specific question. Finally note that data mining aims at discovering "understandable" relationships and summaries that can be interpreted as "novel", "unsuspected", and "useful" knowledge. In other words, the goal of data mining is to offer new insights that are valuable to the data owner.

The algorithms that take data as input and produce models or patterns as output are called *data mining techniques*. To make this thesis self-contained, we summarize several particularly popular data mining techniques in Section 2.3.1. Section 2.3.2 shows how other researchers used data mining to make alarm investigation more efficient. The Sections 2.3.3 and 2.3.4 summarize other, less closely related applications of data mining to intrusion detection.

2.3.1 Classification of Data Mining Techniques

Data mining techniques are algorithms that extract patterns or build models from available datasets. There is no unique and universally accepted classification of data mining techniques. However, the classification shown in Figure 2.2 is consistent with most data mining texts (e.g. [90, 91]). Note that the figure is not exhaustive, and that many more leave nodes could have been added [90, 91].

A preliminary remark on input data representations is in order: Without loss of generality, we assume that the input data is stored in a single relational database table [59]. The columns of this table are called *attributes* and the rows are called *tuples*. Sets of transactions [3] and multidimensional data cubes [2] are other common input data representations in data mining. However, to simplify the presentation, we do not consider these input data representations, here.

At the highest level, data mining techniques are split into predictive and descriptive techniques. *Predictive techniques* use examples, called *training data*, to learn a model that predicts the unknown value of one attribute (e.g. a person's salary) based on the known values of other attributes (e.g. the person's education, age, and employer). In other words, predictive techniques model the relationship between one designated attribute and all the other attributes. *Descriptive techniques*, by contrast, find patterns and summaries that describe a dataset in a human-understandable manner. No training data is needed, and all attributes are treated symmetrically (as opposed to singling out one particular attribute).

Predictive techniques are further sub-classified into regression and classification. In *regression* [53, 118], the attribute being predicted is real-valued, whereas in *classification* [38, 177] it is categorical, i.e. discrete and unordered. The values predicted by classification are referred to as *class labels* to emphasize that they have no numerical meaning. For example, a set of intrusion detection alarms that have been classified into true positives and false positives, can serve as training data to learn a classifier that predicts the class labels ("true positive" versus "false positive") of future, previously unseen alarms. This classifier can be represented by means of if-then rules, decision trees, or some other structure [91, 155].

We next consider the sub-categories of descriptive techniques. *Visual techniques* use colors, shapes, and animations to visually represent data in an easy-to-understand manner [36, 171]. *Association rules* [3, 4, 190] capture implications between attribute values. More formally, association rules have the form $(\wedge_{i=1}^{m}[A_i = v_i]) \implies (\wedge_{i=m+1}^{n}[A_i = v_i])\ [s, c]$, where the A_i ($i = 1, \ldots, n$) are pairwise distinct attributes, the v_i are attribute values, and the numbers $s, c \in \mathbb{R}^+$ (called *support* and *confidence*) are descriptive statistics. This rule is interpreted as: "Database tuples that satisfy the rule's left-hand side have a $c\%$ probability to also satisfy the right-hand side. Moreover, $s\%$ of tuples satisfy both sides."

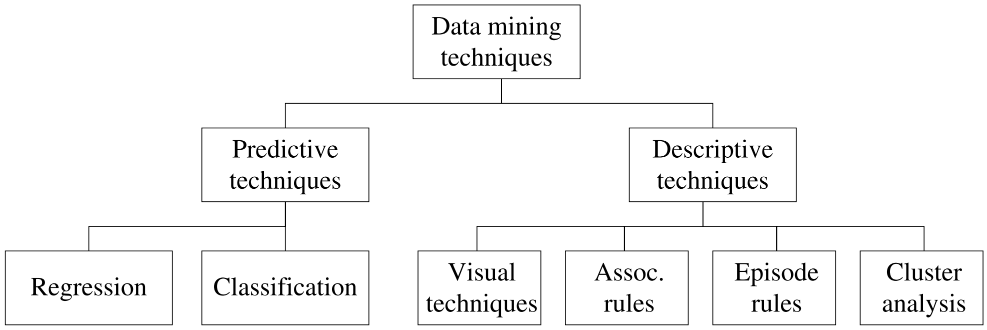


Figure 2.2: A rough classification of data mining techniques.

Note that association rules capture relationships between the attributes of a *single* database tuple. *Episode rules* [142, 143], by contrast, capture relationships between *successive* tuples. This clearly presupposes that tuples are ordered, e.g. by virtue of a time attribute. Assuming that an order exists, episode rules have been defined as implication rules that predict the occurrence of certain tuples based on the occurrence of other tuples. For example, assuming that tuples represent intrusion detection alarms, an episode rule might state that in 50 percent of all cases, an "Authentication failure" alarm is followed within 30 seconds by a "Guest login" alarm. We will formally define episode rules in Section 3.5.

Cluster analysis (a.k.a. *clustering*) [81, 104, 105] seeks to group tuples so that the tuples within a given group/cluster are similar, whereas the tuples of different groups/clusters are dissimilar. Obviously, the notion of similarity is key to this definition. In Euclidean spaces, similarity is relatively easy to define [105]. For example, the inverse of the Euclidean distance is a possible similarity measure. Moreover, this measure captures our intuition that a cluster is an isolated and compact "cloud" in space. In non-Euclidean spaces, categorical attributes such as IP addresses or port numbers significantly complicate the definition of similarity [74, 84]. Chapter 4 has much more to say about cluster analysis. Finally, there are several good textbooks that can be consulted for a broader and more detailed treatment of data mining concepts and techniques [20, 90, 91].

2.3.2 Using Data Mining for Alarm Handling

This section surveys research projects that used data mining to support the investigation of alarms. First, however, recall the distinction between alarms and alarm types: Alarms are the messages that an IDS triggers to report security violations. The alarm type, by contrast, is the attribute that specifies the actual security violation (e.g. SYN flooding, host scanning, buffer overflow, etc.) reported by an alarm. Also recall that $\mathtt{t}[A_i]$ denotes the value that attribute A_i assumes in tuple $\mathtt{t}$.

Manganaris et al. mine association rules to build a "second-level" anomaly detection system that discards "normal" alarms and thereby reduces the work-load of the operator [141]. Implicitly, this work assumes that "normal" alarms are always false positives. The reference model of normal alarm behavior is learned in two steps. First, a time-ordered stream of historical alarms is partitioned into bursts, and second, association rules are mined from these bursts (more on this later). The resulting association rules constitute the reference model of normal alarm behavior. At runtime, alarms are compared against this reference model, and alarms that are consistent with it are considered normal/benign and get discarded.

More precisely, Manganaris et al. model alarms as tuples (t, A), where t is a time-stamp and A is an alarm type. All other alarm attributes such as source IP address, destination IP address, or port numbers are not considered in their model. *Alarm bursts* are sets of alarms that are separated by prolonged alarm-free periods. Internally, alarm bursts are represented as tuples that have one attribute A_i for each possible alarm type A_i. Specifically, let B be an alarm burst, and let $\mathbf{b}$ be its internal representation. Then, $\mathbf{b}[A_i] = 1$ holds if and only if B contains an alarm of type A_i, and $\mathbf{b}[A_i] = 0$, otherwise. Note that this representation suppresses all temporal information about alarms. The tuples representing the alarm bursts are mined for association rules of the form $(\wedge_{i \in I}[A_i = 1]) \Longrightarrow (\wedge_{j \in J}[A_j = 1])$, with disjunct index sets I and J (i.e. $I \cap J = \varnothing$). Association rules about *non-existing* alarm types (e.g. $[A_1 = 1] \Longrightarrow [A_2 = 0]$) are explicitly not searched for.

The set of discovered association rules is used without any inspection or modification as the reference model of normal alarm behavior. At run-time, deviations from this model are detected as follows: The incoming alarm stream is partitioned into alarm bursts, and each alarm burst is considered by itself. Various tests are used to decide whether an alarm burst B is anomalous. For example, suppose the reference model contains the association rule $(\wedge_{i \in I}[A_i = 1]) \Longrightarrow (\wedge_{j \in J}[A_j = 1])$. Moreover, let us assume that alarm burst B contains all the alarm types A_i (i.e. $\forall i \in I : A_i \in B$), but lacks some of the A_j (i.e. $\exists j \in J : A_j \notin B$). Then, alarm burst B is deemed anomalous as it does not contain all A_js, which would have been expected based on the rule $(\wedge_{i \in I}[A_i = 1]) \Longrightarrow (\wedge_{j \in J}[A_j = 1])$. Only anomalous alarm bursts are reported to the operator, while normal ones are discarded.

Note that this approach incurs a potentially high risk of discarding true positives. In fact, the set of discovered association rules is *blindly* used as reference model. As a consequence, association rules that correspond to attacks can creep into the reference model and thereby prevent future detection of these attacks. On the other hand side, given that reference models can contain thousands of association rules [141], it does not seem practical to manually inspect them before usage. The alternative approach of using attack free alarm logs to learn the association rules seems equally difficult to implement [112].

Clifton and Gengo use data mining to find alarm patterns that a human expert can understand and act upon [37]. More precisely, they mine episode rules from historical alarm logs, and use these episode rules to guide the construction of custom-made filtering rules, which automatically discard well-understood false positives. Clifton and Gengo offer hardly any experiments to validate their approach. We therefore replicate their experiments in Section 3.5 and come to the conclusion that episode rules are not particularly suitable in this context. This motivates our work on clustering, which is the main focus of this thesis. Nevertheless, their and our work are obviously related because both mine historical alarm logs in quest for human-understandable patterns.

In the world of telecommunication networks, Klemettinen uses association rules and episode rules to support the development of alarm correlation systems [119]. Hellerstein and Ma pursue the same goal by means of visualization, periodicity analysis, and m-patterns (a variant of association rules requiring mutual implication) [94]. Garofalakis and Rastogi investigate bounded-error lossy compression of network management events [75]. These research projects as well as our own experiments [110, 114] have convinced us that visualization, episode rules, and association rules are too demanding (in terms of time and human expertize) to be used on a larger scale. This has further motivated our work on clustering. To the author's knowledge, the idea of applying clustering to intrusion detection or network management alarms was new at its time of introduction.

2.3.3 The MADAM Approach for Building IDSs

The MADAM ID project at Columbia University [129, 131, 132] is one of the first and best-known data mining projects in intrusion detection. Therefore we briefly survey this project, here. The contents of this section is optional and is not required for an understanding of later chapters.

The MADAM ID project has shown how data mining techniques can be used to construct IDSs in a more systematic and automated manner than by manual knowledge engineering. Specifically, the approach pursued by MADAM ID is to learn classifiers that distinguish between intrusions and normal activities. Unfortunately, classifiers can perform really poorly when they have to rely on attributes that are not predictive of the target concept [130]. Therefore, MADAM ID proposes association rules and episode rules as a means to construct additional, more predictive attributes. In the terminology of MADAM ID, these additional attributes are called *features*.

MADAM ID has been most extensively documented for the case of building network-based misuse detection systems. Therefore, this section also describes MADAM ID in the context of network-based misuse detection. Note, however, that there have been experiments in applying MADAM ID to anomaly detection as well as to host-based misuse detection [129]. Because of space limitations, these experiments are not described here.

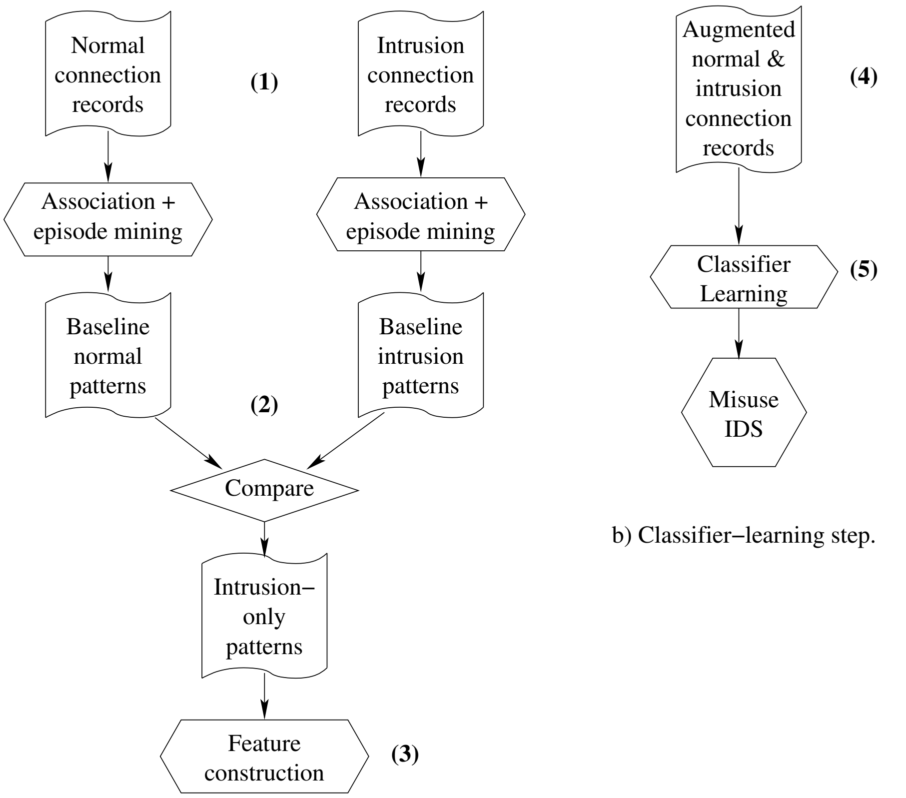
Let us now consider how MADAM ID is used to construct network-based misuse detection systems. The base version of MADAM ID that we discuss here does *not* consider the packet payload of network traffic. Indeed, all network traffic is abstracted to so-called *connection records*. The attributes of connection records store intrinsic connection characteristics such as the source IP, the destination IP, the source and destination ports, the start time, the duration, the header flags, etc. . In the case of TCP/IP networks, connection records summarize TCP sessions.

The most notable characteristic of MADAM ID is that it *learns* a misuse detection model from examples, called *training connection records*. Training connection records are connection records that have been classified into *normal connection records* and *intrusion connection records*. Given a set of training connection records, MADAM ID proceeds in two steps: The *feature-construction step* augments the connection records by additional attributes that are deemed to be relevant for distinguishing intrusions from normal activities. For example, this step might add a new attribute that counts the number of connections that have been initiated during the preceding two seconds to the destination IP address of the current connection record. The feature-construction step is followed by the *classifier-learning step*, which learns a classifier from the augmented training connection records (i.e. the original training connection records extended by the newly constructed attributes). Figure 2.3 illustrates the process in more detail:

(1) The user of MADAM ID procures training connection records and partitions them into normal connection records and the intrusion connection records. MADAM ID offers no support for this step.

(2) Association rules and episode rules are mined separately from the normal connection records and from the intrusion connection records. The resulting patterns are compared, and all patterns that are exclusively contained in the intrusion connection records are collected to form the *intrusion-only patterns*.

(3) The intrusion-only patterns are used in a mostly automatic procedure to derive additional attributes, which are expected to be indicative of intrusive behavior. These additional attributes are counts, averages, and percentages over connection records that share some attribute values with the current connection record.



b) Classifier–learning step.

a) Feature–construction step.

Figure 2.3: Data mining process of building misuse detection systems.

(4) The original training connection records are augmented by the newly constructed attributes.

(5) A classifier is learned that distinguishes normal connection records from intrusion connection records. This classifier — the *misuse IDS* — is the end product of MADAM ID.

It is worth pointing out that the MADAM ID process of Figure 2.3 heavily relies on intrusion detection expert knowledge. For example, expert knowledge is used to prune the number of patterns produced during association and episode rule mining. Also, feature construction is restricted to adding attributes that an intrusion detection expert would consider promising. Moreover, separate classifiers are constructed for different attack classes, and these classifiers are subsequently combined. Finally, domain knowledge is used to manually define suitable features that summarize the packet payload. These more advanced aspects of MADAM ID are described elsewhere [129, 131, 132].

2.3.4 Further Reading

In this section, we briefly survey other related work that applied data mining to intrusion detection. A more detailed overview of the field can be found in a recent book edited by Barbará and Jajodia [17].

The vast majority of researchers have used data mining to build non-parametric models for anomaly detection. This is very similar to early work in intrusion detection, except that normal behavior is represented by models based on data mining rather than statistics [109, 188]. For example, Wisdom & Sense is an anomaly detection system that mines association rules from historical audit data to represent normal behavior [198]. Similarly, Teng et al. use a form of automatically learned episode rules to represent normal user behavior [197]. The idea of Lankewicz and Benard is to cluster audit log records and to represent each cluster by a single "typical" audit log record. These typical audit log records form the model of normal behavior against which future audit log records are compared [126]. A similar idea has been pursued by Lane and Brodley, who cluster attack-free shell command sequences and define the "cluster centers" to represent normal behavior. Subsequently, anomalous command sequences can be detected based on their distance to the cluster centers [125]. Portnoy et al. cluster historical connection records and label large clusters as "normal" and small clusters as "abnormal" [173]. At run-time, new connection records are classified according to their distance to the closest labeled cluster. Barbará et al. use incremental association rule mining to detect anomalous network traffic patterns in real-time [18]. Finally, there is a long list of research projects that have tried to model system call sequences by a variety of different models, including neural networks, hidden Markov models, as well as fixed and variable length patterns. The work by Warrender et al. [201] and Debar et al. [46] is representative of this research.

Mukkamala et al. present work that does not fall in the category "data-mining-based anomaly detection". Their idea is to use data mining techniques to reduce the amount of audit data that needs to be maintained and analyzed for intrusion detection [158]. Lam et al. report similar work in audit data reduction [123].

2.4 Alarm Correlation

Alarm Correlation Systems (ACSs) [40, 41, 44, 48, 191, 199] post-process intrusion detection alarms in real-time and automate part of the alarm investigation process. More precisely, ACSs attempt to group alarms so that the alarms of the same group pertain to the same phenomenon (e.g. the same attack). Then, only the alarm groups are forwarded to the human operator. In that way, ACSs offer a more condensed view on the security issues raised by an IDS. In addition, they make it easier to distinguish real security threats from false positives.

ACSs are clearly related to this thesis work because they address the same problem. Moreover, they pursue a very similar approach, which could be called "real-time clustering" (after all, ACSs group/cluster alarms in real-time). Nevertheless, the following considerations show the need for the new off-line clustering method presented in this dissertation:

Depth of analysis: Due to harsh real-time requirements, ACSs can only perform a limited amount of analysis. For example, consider a phenomenon that only occurs on Saturdays (e.g. false positives due to weekly system backups). Our off-line clustering method is able to correctly group and report the resulting alarms, whereas ACSs do not have this capability because it is difficult to implement in real-time. Moreover, to reliably identify a weekly alarm pattern, one must observe at least several weeks of alarms. Clearly, delaying correlation results for weeks defeats the very purpose of real-time alarm correlation. Similarly, processing free-text alarm attributes (e.g. the context, which stores raw audit records, cf. Section 1.4) is costly and is consequently not done by real-time ACSs. By contrast, our off-line clustering method analyzes free-text attributes and thereby improves its results.

Bias: ACSs are generally optimized to find alarm groups that result from *attacks*. This attack-centric bias has far-reaching consequences. For example, some ACSs reassess the severity of alarm groups and discard alarm groups that are deemed benign [48, 191]. Other ACSs use complex techniques to deal with spoofed source IP addresses, multi-stage attack scenarios, or stealthy attacks [41, 44, 199]. Moreover, the publications on ACSs use exclusively attacks to validate their systems. This attack-centric bias contrasts with our goal of finding *any* large group of systematically generated alarms, so that we can identify and remove its root cause. Many of these alarm groups are not the result of attacks, and today's ACSs are not particularly suitable for finding them. Our off-line clustering method, by contrast, is biased in a way that favors the discovery of such alarm groups.

Ease of use: Today's ACSs are difficult to configure. For example, some ACSs have dozens of configuration parameters, which take experience to set [48, 199]. Other ACSs face a knowledge engineering bottleneck because they require the user to specify correlation rules [40, 41]. The ACS in [44] learns correlation rules from the user. To this end, the user has to manually correlate alarms, so that the system can learn his or her ability. Clearly, manual alarm correlation is difficult and error-prone. As will become apparent throughout this thesis, our clustering method has none of these drawbacks, and is easy and intuitive to use.

The remainder of this section surveys three recent alarm correlation systems. The presentation of these systems uses the alarm model of Section 1.3, i.e. alarms are modeled as tuples consisting of alarm attributes and their corresponding values. Moreover, $\mathbf{a}[A_i]$ denotes the value that attribute A_i assumes in alarm $\mathbf{a}$.

2.4.1 Tivoli Aggregation and Correlation Component

The Tivoli Aggregation and Correlation Component (TACC) [48] cascades three processing steps, through which each alarm is pipelined in strict order. The first step unifies different data representations, e.g. by mapping between host names and IP addresses or between port numbers and service names. The second step uses expert system rules to merge duplicate alarms and to pinpoint faulty IDSs. For example, suppose a particular attack is *always* reported by two independent IDSs. Then, an expert system rule can be written to merge the resulting duplicates. Moreover, a second expert system rule can be used to issue a warning message when only *one* of the two IDSs reports said attack. Clearly, this indicates that the other IDS is probably broken. The third step of the TACC groups alarms that agree in one or more alarm attributes. Moreover, it counts the number of alarms per group and time window, and raises a meta-alarm if this count exceeds a user-defined threshold value.

Let us consider the third step in more detail. To begin with, the TACC defines three so-called *aggregation axes*, namely the source IP address, the destination IP address, and the alarm class, which generalizes the alarm type into broader categories. Each combination of aggregation axes defines one out of seven ways of grouping alarms. For example, the combination $\{Src\text{–}IP, Dst\text{–}IP\}$ specifies to group alarms that originate from the same source IP address and go to the same destination IP address. Similarly, the combination $\{Alarm\text{–}class\}$ specifies to group alarms that share the same alarm class. Note that all alarms within a given alarm group have identical values along the group's aggregation axes.

Now let us consider how a newly triggered alarm is processed. This alarm is simultaneously added to all seven groups, i.e. *(1)* to the group of alarms that have the same source IP address as the new alarm, *(2)* to the group of alarms that have the same destination IP address as the new alarm, ..., *(7)* to the group of alarms that have the same source IP address, destination IP address, and alarm class as the new alarm. For each of these seven groups, a separate sliding time window is maintained and the number of alarms within this time window is counted. If (and only if) this count exceeds a user-defined threshold value, then a meta-alarm is raised. Note that an alarm group whose count does not exceed the threshold value is not reported to the user!

The TACC approximates sliding time windows by means of a weighted sum scheme, which is particularly resource-sparing. More precisely, each time a new alarm is added to a group, the new count value is calculated according to the formula $count_{new} = 1 + count_{old} \times 2^{\tau \times (t_{old} - t_{new})}$. Here, t_{new} and t_{old} are the time-stamps of the new and the preceding alarm, respectively. The parameter $\tau \in \mathbb{R}^+$ is a user-defined fading factor, also referred to as half-life. For example, if τ is chosen to be $^1/_{100}$, then the weight of the old count value fades to half its nominal value if 100 seconds elapse before the next alarm occurs.

2.4.2 Probabilistic Alert Correlation

The Probabilistic Alert Correlation (PAC) system [199] represents alarm groups by means of so-called *meta-alarms*. To understand meta-alarms, first note that PAC supports set-valued attributes. Using this feature, a group of alarms can be represented by a single meta-alarm that is obtained by taking attribute by attribute the union of attribute values. For example, the alarm group $\{(1, A, \alpha), (2, B, \alpha), (3, A, \alpha)\}$ would be represented by the meta-alarm $(\{1, 2, 3\}, \{A, B\}, \{\alpha\})$. Clearly, the mapping of alarm groups to meta-alarms is not injective.

The PAC system maintains a continuously updated collection of meta-alarms. Whenever the PAC system receives a new alarm, it compares the alarm to all existing meta-alarms and calculates the similarities between the new alarm and the various meta-alarms. The new alarm is then merged with the most similar meta-alarm, provided the similarity exceeds a user-defined minimum similarity threshold. By contrast, an alarm that lacks minimum similarity to all existing meta-alarms starts a new meta-alarm of its own.

Very roughly, the similarity between an alarm and a meta-alarm is defined as the weighted sum of attribute-wise similarities. Towards a more precise definition, let us assume that alarms have n attributes $A_1, \ldots, A_n$. Let $t_1, \ldots, t_n \in [0, 1]$ be attribute-wise minimum similarity thresholds, let $w_1, \ldots, w_n \in [0, 1]$ be attribute-wise weights (so-called expected similarities), and let $s_1(\cdot, \cdot), \ldots, s_n(\cdot, \cdot)$ be attribute-wise similarity functions that return a value between 0 (denoting complete dissimilarity) and 1 (denoting identity). The parameters t_i, w_i, and $s_i(\cdot, \cdot)$ will be discussed in a moment. First, however, we define the similarity $sim(\mathbf{a}, \mathbf{m})$ between an alarm $\mathbf{a}$ and a meta-alarm $\mathbf{m}$:

$$sim(\mathbf{a}, \mathbf{m}) := \begin{cases} 0, & \text{if } s_i(\mathbf{a}[A_i], \mathbf{m}[A_i]) < t_i \text{ for any } i \\ \frac{\sum_{i=1}^{n} w_i \cdot s_i(\mathbf{a}[A_i], \mathbf{m}[A_i])}{\sum_{i=1}^{n} w_i}, & \text{otherwise.} \end{cases} \tag{2.2}$$

The PAC system provides little guidance for setting the parameters t_i and w_i, $i = 1, \ldots, n$. Moreover, some of these parameters are situation-specific, which further complicates the task of finding reasonable values. For example, the minimum similarity threshold t_{Src-IP} of the source IP address is situation-specific. Specifically, t_{Src-IP} should be close to 0 for alarms that are prone to source address spoofing (e.g. SYN flooding alarms). On the other hand side, t_{Src-IP} should be much higher when source address spoofing is unlikely.

The PAC system suggests intuitively appealing but ultimately ad hoc similarity functions $s_1(\cdot,\cdot), \ldots, s_n(\cdot,\cdot)$. For example, two destination port lists are deemed more or less similar based on their degree of overlap. For alarm types, PAC uses a similarity matrix with values of unity along the diagonal and off-diagonal values that heuristically express similarity between the corresponding alarm types. Time similarity is a step function that drops after one hour from 1 to 0.5, and the similarity between IP addresses considers if the two IP addresses come from the same subnet.

The PAC system is monolithic and cannot easily be distributed. However, distributed processing is still possible by recursively applying PAC to its own results. Specifically, the authors of PAC use PAC to correlate individual alarms into *threads*, threads into *security incidents*, and security incidents into *attack reports*. The very same PAC system is used on all three levels of the correlation hierarchy. However, the respective parameter values differ depending on the level (see [199] for details). Note that the three-layered correlation hierarchy comes at the cost of tripling the number of parameters that need to be set!

2.4.3 Alert Stream Fusion

The Alert Stream Fusion (ASF) system [44] maintains a continuously updated collection of alarm groups, called *scenarios*. Whenever the ASF system receives a new alarm from an IDS, it compares the alarm to all existing scenarios and calculates the probabilities that the new alarm belongs to the respective scenarios. The new alarm is then assigned to the scenario that produces the highest probability score. If all probability scores are below a user-defined threshold then the new alarm is not added to any scenario, but rather starts a new scenario of its own. The assignment of an alarm to a scenario is final and irreversible.

The probability that a new alarm $\mathbf{a}$ belongs to a scenario S is a function of *only* the new alarm $\mathbf{a}$ and the most recent alarm in S. All the other alarms in S are not considered. The function that calculates the probability scores is learned by means of predictive data mining techniques from labeled training data. The original article by Dain and Cunningham [44] compares different predictive data mining techniques with respect to their ability to learn “good” probability functions. We here focus on decision trees, which produced the best results.

The training data needed by the decision tree algorithm is obtained by manually correlating historical alarms. To this end, one has to manually execute the program of Figure 2.4. This yields positive training examples, which have the class label $merge$ (cf. line 5 of Figure 2.4), as well as negative training examples, which have the class label $\neg merge$ (cf. line 6 of Figure 2.4). Using these training examples, it is possible to learn a decision tree. This decision tree is subsequently used to predict the $merge$ and $\neg merge$ decisions for new, previously unseen alarms. It has been noted that the manual correlation of historical alarms is labor-intensive and site-dependent [44].

It is instructive to compare the ASF system to the PAC system of Section 2.4.2: First, meta-alarms in PAC clearly correspond to scenarios in ASF, and similarities correspond to probabilities. Second, the calculation of similarities in PAC relies on heuristics and human expertise, while ASF uses predictive data mining techniques to learn a suitable probability measure. Third, meta-alarms summarize *all* alarms in an alarm group. As a consequence, the similarity between a new alarm and a meta-alarm implicitly takes *all* alarms in the respective alarm group into account. ASF, by contrast, compares a new alarm only to the most recent alarm of a scenario, while all the other alarms in the scenario are ignored.

Input: A chronologically ordered sequence $\mathbf{a_1}, \ldots, \mathbf{a_n}$ of alarms;
Output: Training data for decision tree learning;
Algorithm:
1: ***for*** $i := 1$ ***to*** n ***do*** { // *Loop over all alarms* $\mathbf{a_1}, \ldots, \mathbf{a_n}$.
2: ***for each*** scenario S in memory ***do*** {
3: Let $\mathbf{l}$ be the last alarm that was added to S;
4: ***if,*** based on human judgment, $\mathbf{a_i}$ belongs to S
5: ***then*** add $\mathbf{a_i}$ to S and output the triple $(\mathbf{a_i}, \mathbf{l}, \mathit{class\_label} = \mathit{merge})$;
6: ***else*** output the triple $(\mathbf{a_i}, \mathbf{l}, \mathit{class\_label} = \neg \mathit{merge})$;
7: }
8: ***if*** $\mathbf{a_i}$ belongs to none of the existing scenarios S
9: ***then*** create a new scenario S' and add $\mathbf{a_i}$ to S';
10: }

Figure 2.4: Algorithm for manually deriving training data from historical alarms.

Chapter 3

Using Data Mining for Root Cause Analysis

As explained in the introduction, this thesis uses data mining to extract alarm patterns that a skilled user can interpret in terms of root causes. The present chapter offers a more formal treatment of this idea. Specifically, Section 3.1 revises the definition of root causes and makes it more precise. Section 3.2 explains how data mining can be used to support the discovery of root causes. Moreover, the limitations of data mining in this context are discussed. Section 3.3 characterizes the kind of alarm patterns that root causes typically generate, and explains that a data mining technique should ideally discover these alarm patterns. The discussion in this section is purely theoretical, and we therefore experimentally validate it in Section 3.4. Section 3.5, finally, summarizes our experience with episode rules, a data mining technique that has significant intuitive appeal for root cause analysis.

3.1 Root Causes and Root Cause Analysis

This section revises our earlier definitions of root cause and root cause analysis. In addition, the genesis of root causes is studied.

3.1.1 Revised Definitions

Section 2.2.1 has offered a generic definition of root causes. Nonetheless, it is desirable to find a more specific definition, which is tailored to intrusion detection and which provides stronger guidance to the people performing the root cause analysis. Earlier work in the dependability field [127], lays the foundation for such a revised definition.

Without embarking on a general discussion of dependability concepts, we briefly show how these concepts apply to computer security [174]. To begin with, the dependability community defines an *information system* as a collection of interacting *components* such as hosts, printers, routers, LANs, etc. . A *security failure* occurs when the users of the information system notice a loss of confidentiality, integrity, or availability. A corrupted or compromised component that may lead to a subsequent security failure is called an *error*. The adjudged or hypothesized cause of an error is a *fault*. Note that error and security failure are distinct concepts. Specifically, a security failure occurs when an error "passes through" the system-user interface and affects the service as perceived by the user. Finally, IDSs perform *error detection*, and false positives are alarms that incorrectly rate a component as erroneous.

For example, consider an information system that the user perceives as a single black box. Moreover, assume that one component of the information system is compromised by an attacker. This compromised component is an error, and a successful attack in combination with the exploited vulnerability are the faults that caused this error. The error remains invisible to the user as long as the attacker keeps still. On the other hand side, system failure occurs when the attacker degrades the system's confidentiality, integrity, or availability to the point where it becomes apparent to the user. Intrusion detection systems detect erroneous components when they start to behave abnormally, or when they attack other components. Therefore, intrusion detection is a form of error detection.

The above dependability definitions are too tight for our purposes. Specifically, an error was defined as a corrupted or compromised component, and a fault was defined as the cause of an error. This thesis, however, is particularly interested in the causes of perceived errors. A *perceived error* is a component that triggers alarms, no matter whether these alarms are true or false positives. Thus, an actually compromised component as well as a healthy component that only the IDS believes to be compromised are both perceived errors. To avoid clashes with dependability terminology, we use the term *root cause* rather than *fault* to refer to the cause of a perceived error. More precisely:

Definition 3.1 *A* **root cause** *is a problem that affects one or more components and causes them to trigger alarms.* **Root cause analysis** *is the task of identifying root causes as well as the components affected by them.* □

Note that root causes are detached from the components they affect. Thus, a root cause (e.g. a worm) affects one or more components (e.g. the hosts in a subnet), which in turn causes these components to trigger alarms (when the worm spreads by attacking other machines). Similarly, a non-standard protocol can affect a server and cause it to trigger "Suspicious protocol" alarms. Root cause analysis is concerned with identifying the type and locations of root causes.

The above definitions might not be as rigorous as one had hoped. However, by linking the definition of root causes to the dependability concept of faults, one can justifiably argue that there probably is no better definition. In fact, the dependability community has used the notion of faults for over a decade without suggesting a formal definition. Therefore, it is unlikely that there is a formal definition for the intrusion detection equivalent of faults.

3.1.2 Genesis of Root Causes

The *genesis* of a root cause describes how it came into existence. In other words, the genesis of a root cause describes the forces that introduced it. Studying the genesis of root causes is important because it promotes a better understanding of what root causes are and how/why they come into existence. It must be emphasized that this section does *not* attempt to classify root causes. Classifying root causes is an extremely difficult task. In fact, attacks are only one type of root causes, but even the problem of attack classification is highly controversial [99, 121, 124]. Therefore, no attempt to classify root causes is made.

Figure 3.1 shows that root causes can be introduced intentionally or inadvertently. A root cause is *intentional* if a human being introduces it consciously and on purpose. By contrast, a root cause is *inadvertent* if it is the unconscious byproduct of some action. The distinction between intentional and inadvertent root causes is subtle. For example, a malicious system administrator might intentionally install a Trojan horse, or a thoughtless user might inadvertently install it along with other software that she downloaded from the Internet. In both cases, the end effect is the same, but the genesis of the Trojan horse (which is the root cause in this example) is different.

Intentional root causes can be introduced with *malicious* intent or with *non-malicious* intent. Malicious intent means that somebody brought the root cause into existence, intentionally so that it would compromise system security. Examples of malicious root causes are worms and the execution of attack scripts. Non-malicious root causes, on the other hand side, were introduced for the desirable functionality they provide. Desirable functionality can consist in new *protocols & services* that a system administrator installs to satisfy the user community. However, some IDSs trigger alarms when they observe unknown protocols or exploitable services. Similarly, changes in the *topology & architecture* can cause suspicious traffic patterns, which trigger alarms on many network-based IDSs.

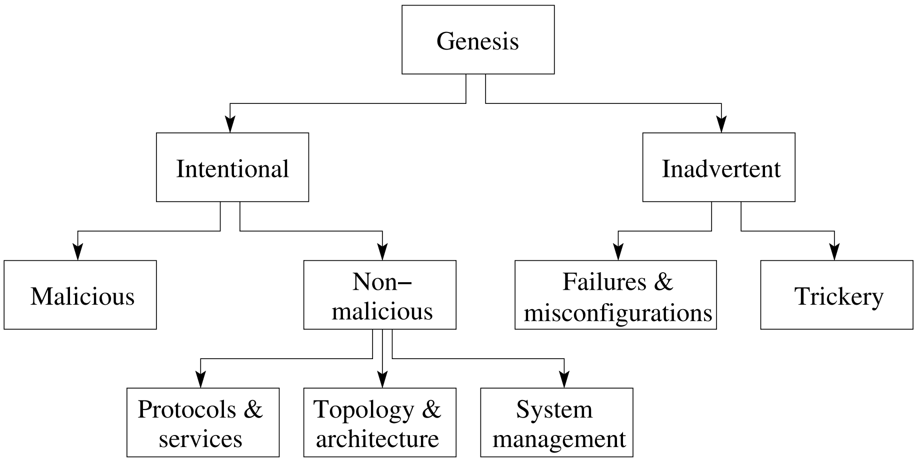


Figure 3.1: The genesis of root causes, or how root causes enter a system.

Finally, *system management* comprises certain highly desirable tasks such as vulnerability scanning, network mapping, are-you-alive messages, and the querying of potentially sensitive MIB variables. All of these tasks can and frequently do trigger intrusion detection alarms.

Root causes that were introduced inadvertently can result from *failures & misconfigurations* or from *trickery*. An example of a failure is the broken TCP/IP stack of Chapter 1. A thoughtless user that launches an e-mail attachment and thereby infects his machine with a virus is an example of a root cause that was inadvertently introduced by trickery. Section 3.3.1 offers more examples of root causes. Section 3.3.2 uses the results of this section to gain some general insights into the alarm patterns that root causes induce.

3.2 Conceptual Problem Description

The present section lays the methodological foundation of this thesis. Specifically, Section 3.2.1 explains how data mining can be used to support root cause analysis, and Section 3.2.2 specifies the algorithmic requirements that a data mining technique should satisfy to be of maximal value for root cause analysis.

3.2.1 The Role of Data Mining in Root Cause Analysis

In this work, the role of data mining is to group alarms that have the same root cause so that a human expert can interpret the resulting alarm groups in terms of root causes. The entity relationship diagram of Figure 3.2 describes

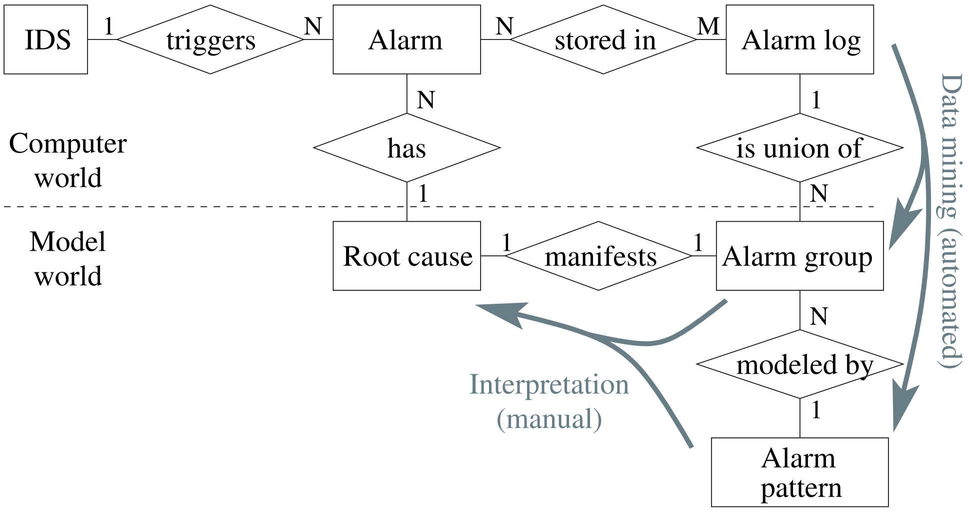


Figure 3.2: Entity relationship diagram of key concepts.

the relationship between alarms, root causes, and data mining in a more formal manner. The dashed line in the figure separates computer world concepts (which have an actual existence in the form of hardware components or memory state) from model world concepts (which only exist in the world of our thinking). The entities and relationships are as follows:

- An IDS triggers zero or more alarms $\mathbf{a_i}$, $i = 1, 2, 3, \ldots$.
- Each alarm $\mathbf{a_i}$ has a unique root cause $rc(\mathbf{a_i})$, which is the reason of its existence. Alarms are stored in alarm logs. (Note that each alarm must have a root cause because otherwise it would not exist. Moreover, according to Definition 3.1, each alarm is caused by a *single* problem — its root cause. Even though this problem might be the conjunction of multiple sub-problems, we conceptually treat it as the single root cause.)
- An *alarm log* is a multi-set (a.k.a. bag or collection) of alarms. Modeling alarm logs as multi-sets is correct because alarms are implicitly ordered by virtue of the time-stamp attribute. We use pointed brackets for multi-sets (e.g. $L_1 = <\mathbf{a}, \mathbf{a}, \mathbf{b}>$, $L_2 = <\mathbf{b}, \mathbf{c}>$) and re-define the set operators so that they take multiple occurrences of the same element into account (e.g. $|L_1| = 3$, $|L_2| = 2$, $L_1 \cup L_2 = <\mathbf{a}, \mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}>$, $L_1 \not\subseteq L_2$, $<\mathbf{a}, \mathbf{a}> \subseteq L_1$, etc.).
- An *alarm group* is a multi-set of alarms that share the same root cause. The root cause *RC* manifests itself in the alarm group $<\mathbf{a_i} \mid rc(\mathbf{a_i}) = RC>$.
- The alarm log is the union of all alarm groups. Another way to look at this is that the alarm groups are the result of partitioning the alarm log along the boundaries of root causes.

- An alarm group is modeled by an *alarm pattern.* An alarm pattern is an expression in some formal language that characterizes or summarizes the alarms of an alarm group. Note that an alarm pattern is an abstraction of the alarm group it models. Therefore, it can include alarms that are not in the alarm group, and, conversely, it can exclude some alarms that the alarm group actually contains. The principal requirement is that an alarm pattern must model the main characteristics of an alarm group.

The gray arrows of Figure 3.2 specify a *normative model* for how root cause analysis *should* be done (see below for the practical infeasibility of this model). According to this model, data mining partitions the alarm log into alarm groups, and derives alarm patterns for the alarm groups. The alarm groups and patterns are subsequently presented to a human expert who is tasked to interpret them in terms of root causes. This interpretation task is vastly simplified by the availability of alarm patterns. In fact, Section 3.3 will show that alarm groups can be large and difficult to comprehend. Therefore, listing their constituent alarms can easily overwhelm the human expert with a vast amount of information that is hard to make sense of. Alarm patterns mitigate this problem by succinctly summarizing alarm groups.

The practical infeasibility of the data mining step of Figure 3.2 results from the requirement that all alarms of an alarm group must share the same root cause. The data mining step cannot enforce this constraint because root causes are a model world construct that data mining is not aware of. To illustrate this, let us consider a machine whose broken TCP/IP stack fragments most IP traffic. Suppose that this machine is behind a router that itself fragments a substantial fraction of the traffic passing through it. Now, let an IDS in front of the router trigger a "Fragmented IP" alarm for a packet from said machine. Unless substantial knowledge about the state of the system is available, there is no way of deciding if the alarm's root cause is the broken TCP/IP stack or the fragmenting router. More complex scenarios are conceivable. In summary, if only an alarm log is given, then it is not possible to decide whether two or more alarms have the same root cause.

Given the infeasibility of constructing alarm groups, we do not scrap the root cause analysis framework of Figure 3.2, but instead accept that data mining can merely approximate alarm groups and their corresponding alarm patterns. Clearly, a data mining technique is all the more useful, the closer it comes to being correct and complete in the following sense:

Definition 3.2 *In the context of this thesis, we call a data mining technique* **correct** *if it is guaranteed to return nothing but alarm groups, i.e. multi-sets whose constituent alarms share the same root cause. A data mining technique is called* **complete** *if alarms of the same root cause are never assigned to separate multi-sets.* □

Intuitively, correctness means that a data mining technique reduces redundancy by grouping alarms that share the same root cause. Incorrect data mining techniques are undesirable because they can mix alarms of different root causes. That makes it harder and possibly even misleading to interpret the data mining results. Completeness without correctness enforces that alarms of the same root cause are always bundled together (but possibly with further alarms of different root causes). Completeness in combination with correctness guarantees that maximal alarm groups are found, i.e. alarm groups that cannot be grown any further without violating the correctness property. Maximal alarm groups are desirable because they eliminate as much redundancy as possible.

A correct and complete data mining technique partitions the alarm log along root cause boundaries. As previously explained, such a data mining technique does not exist. A correct but totally incomplete data mining technique performs no grouping at all, and returns a separate alarm group for each alarm. Conversely, a complete but totally incorrect data mining technique returns a single multi-set, namely the alarm log itself. Both alternatives are useless. In general, the more aggressively a data mining technique groups alarms, the better its chances to be complete, and the higher its risk to become incorrect. Finding a data mining technique that balances correctness versus completeness is the topic of Section 3.3.

3.2.2 Algorithmic Data Mining Requirements

A priori, the root cause analysis approach of the last section does not stipulate any particular technique for the data mining step. Clearly, a prospective data mining technique should rate high with respect to correctness and completeness. In addition, this section identifies five requirements that a data mining technique should satisfy to be of maximal value to root cause analysis:

Scalability: IDSs can trigger well over a million alarms per month (cf. the column "Max" of Table 1.2, which indicates for the year 2001 the maximum number of alarms per month). Therefore, scalability in the size of the alarm log is an important requirement.

Noise tolerance: Alarm logs can be very noisy in the sense that they contain bizarre, spurious, and almost unexplainable alarms [19, 168]. A prospective data mining technique must tolerate this kind of noise.

Ease of use: It has been observed that many data mining techniques require years of experience to be used correctly and effectively [82, 189]. Moreover, improper use was associated to meaningless or even misleading results [9, 90, 105]. It is therefore desirable for a data mining technique to be easy and intuitive to use. This is particularly true as the people performing root cause analysis are most likely to be security rather than data mining experts.

Multiple attribute types: As explained in Section 1.4, intrusion detection alarms can contain numerical attributes (e.g. counts and size attributes), categorical attributes (e.g. port numbers and IP addresses), time attributes, and free-text attributes (e.g. raw audit records). Ideally, a data mining technique should support and use all of these attribute types.

Interpretability & relevance of results: The data mining step should only generate highly interpretable, relevant, and non-redundant results. This is important because the alarm patterns are interpreted by a human expert. Hence, to minimize the risk of misinterpretations and to limit the human cost of working with the alarm patterns, it is essential for them to be of high quality.

The scalability, ease of use, and multiple attribute type requirements rank among the classic research challenges in data mining [82, 189]. Furthermore, it has been observed that many existing data mining techniques tend to generate large numbers of obvious or irrelevant patterns [28, 120, 137, 187]. This shows how demanding the above requirements are, and how challenging it is to find a data mining technique that is suitable for root cause analysis.

3.3 Approximation of Alarm Groups

The last section has shown that it is not possible to implement a correct and complete data mining technique for root cause analysis. Therefore, the present section investigates the next best alternative, namely the approximation of alarm groups. For the time being, our goal is to specify a "good" approximation technique, without regard to the algorithmic requirements of Section 3.2.2. The algorithmic requirements and the issue of implementation will be considered in Chapter 4.

To find a data mining technique that is good at approximating alarm groups, it is necessary to understand how data mining techniques work. Data mining techniques assume that the dataset at hand contains certain patterns, and they search for instances of these patterns. The patterns assumed are specific to each individual technique. For example, episode rule mining [142, 143] assumes sequential patterns, periodicity mining [94] assumes periodic patterns, and linear regression [53, 118] assumes that the dataset follows a linear pattern. Accordingly, these data mining techniques search for sequential, periodic, and linear patterns, respectively. Because patterns have a characteristic structure, it is relatively easy for a data mining technique to spot their occurrences.

In other words, data mining techniques perform pattern extraction. Applied to Figure 3.2, this means that data mining techniques discover alarm patterns, *not* alarm groups. The alarm groups must be approximated after the fact by determining for each alarm pattern the multi-set of alarms that match it. As a consequence, the question of which data mining technique is best at approximating alarm groups is ill-posed, because data mining techniques are unaware of alarm groups. Towards a better question, let us define that a root cause *induces* an alarm pattern if, in fact, the root cause manifests itself in an alarm group that can be modeled by said alarm pattern (cf. Figure 3.2). Then, the right question is: What are the alarm patterns that root causes typically induce, and which data mining technique is best at finding them? The present section addresses this question.

By way of illustration, suppose that root causes induced periodic alarm patterns. For the purpose of root cause analysis, we would then use a data mining technique that extracts periodic patterns. Our implicit assumption is that such a technique is likely to group alarms that actually share the same root cause. Even though this assumption is not always correct, it is typical of the kind of assumptions made in signature-based intrusion detection and, more generally, in abductive inference (cf. Section 2.2.2). For example, attack signatures are generally derived by arguing that if an attack has the manifestation M, then detection of M implies the attack. Analogously, we argue that if root causes typically induce alarm patterns of class C, then detection of an alarm pattern from class C is indicative of a root cause.

Section 3.3.1 studies ten representative root causes to understand the alarm patterns that they induce, and Section 3.3.2 generalizes the results obtained. The goal of both sections is to understand the alarm patterns that *typical* root causes induce. No attempt is made to characterize the alarm patterns that any arbitrary root cause could possibly induce. In fact, such an attempt would be futile. However, as will become apparent in later sections, there is significant practical value in understanding the alarm patterns that typical root causes induce.

3.3.1 Some Representative Examples

This section uses examples to illustrate root causes and the alarm patterns they induce. The sample root causes considered are:

1. A HTTP server with a broken TCP/IP stack that fragments outgoing traffic. Clearly, “Fragmented IP” alarms ensue when the server responds to clients requests.
2. A misconfigured secondary DNS server, which does half-hourly DNS zone transfers from its primary DNS server. The resulting “DNS Zone Transfer” alarms are no surprise.

3. A Real Audio server whose traffic remotely resembles TCP hijacking attacks. This caused our commercial IDS to trigger countless “TCP Hijacking” alarms.
4. A firewall that has Network Address Translation (NAT) enabled funnels the traffic of many users and thereby occasionally seems to perform host scans. In detail, a NAT-enabled firewall acts as proxy for its users. When these users *simultaneously* request external services, then the firewall proxies these requests and the resulting SYN packets resemble SYN host sweeps.
5. A load balancing reverse proxy such as Cisco LocalDirector that dispatches Web client requests to the least busy server. The resulting traffic patterns resemble host scans that trigger alarms on most IDSs.
6. A network management tool querying sensitive MIB variables triggers alarms on most IDSs. (Other network management tasks such as vulnerability scanning or network mapping offer further examples of root causes.)
7. Macintosh FTP clients, which issue the SYST command on every FTP connection, trigger an abundance of “FTP SYST command” alarms. The FTP SYST command is reported by some IDSs because it provides reconnaissance information about the FTP server.
8. An attacker running a brute-force password guessing attack against our Telnet server.
9. A distributed denial-of-service (DDoS) attack [51] being launched from an external network against a Web hosting site triggered “SYN Flooding“ alarms.
10. External Code Red infected machines [33] scanning the internal network for vulnerable servers.

Note that the alarms of the first root cause originate from source port 80 of the HTTP server. Moreover, all of these alarms are targeted at HTTP clients on non-privileged ports. In addition, these alarms always have “Fragmented IP” as alarm type. Therefore, the alarm pattern induced by the first root cause can be represented as shown in the first row of Table 3.1. (For the sake of brevity, we do not include time-stamps and the context attribute in Table 3.1.) Similarly, the second row of the table shows that the second root cause induces the alarm pattern “’DNS zone transfer’ alarms being triggered from a non-privileged port of the secondary DNS server against port 53 of the primary DNS server“. Analogously, the remaining rows of Table 3.1 show the alarm patterns that the other root causes induce. The “RC” (root cause) column of the table refers to the item numbers in the above enumeration of root causes; the entry “Non-priv.” denotes the set $\{1025, \ldots, 65535\}$ of non-privileged ports, and the entry “Privileged” stands for the set of privileged ports below 1025.

Table 3.1: The alarm patterns induced by ten sample root causes.

RC	Source IP	Src-Port	Destination IP	Dst-Port	Alarm Type
1	HTTP server	80	HTTP clients	Non-priv.	Fragmented IP
2	Sec. DNS server	Non-priv.	Prim. DNS server	53	DNS zone transfer
3	Real Audit server	7070	Real Audio clients	Non-priv.	TCP hijacking
4	Firewall	Non-priv.	External network	Privileged	Host scan
5	Reverse proxy	Non-priv.	HTTP servers	80	Host scan
6	Mgmt. console	Non-priv.	SNMP clients	161	Suspicious GET
7	Mac FTP clients	Non-priv.	FTP server	21	FTP SYST
8	Attacker	Non-priv.	Telnet server	23	Password guessing
9	External network	Non-priv.	HTTP servers	80	SYN flood
10	External network	Non-priv.	Internal network	80	Code Red

The rows of Table 3.1 are called *generalized alarms*. Generalized alarms are similar to ordinary alarms, with the exception that attribute values may be generalized. A *generalized attribute value*, such as “Non-priv.” or “Internal network”, is a concept name that represents a set of elementary attribute values. For example, “Non-priv.” represents the set $\{1025, \ldots, 65535\}$ of non-privileged ports. Based on the above examples, we postulate that many root causes manifest themselves in alarm groups that are adequately modeled by generalized alarms. By *adequately*, we mean that generalized alarms are capable of capturing and representing the main features of alarm groups. In other words, little information is lost when modeling alarm groups by generalized alarms.

Another important observation is that most root causes of Table 3.1 are extremely persistent in the sense that they keep generating alarms until someone removes them. As a consequence, these root causes are likely to manifest themselves in *large* alarm groups. For example, the first root cause triggers a “Fragmented IP” alarm whenever the HTTP server responds to a client request. Typically, HTTP servers are heavily used, and consequently, “Fragmented IP” alarms abound. Similarly, the misconfigured secondary DNS server triggers one “DNS zone transfer” alarm every thirty minutes. This makes 1440 alarms a month. Using analogous arguments, it becomes clear that all of the above root causes can be expected to trigger large amounts of alarms. To summarize:

Proposition 3.1 (Alarm cluster hypothesis) *Root causes frequently manifest themselves in* large *alarm groups that are* adequately *modeled by* generalized alarms. □

As explained in Section 2.3.1, a cluster is a group of *similar* objects. Therefore, an alarm group that is adequately modeled by a generalized alarm is a *cluster*.

This follows because the alarms of such an alarm group are mutually similar in the sense that they are subsumed by the same (adequate) generalized alarm. This explains the origin of the name “alarm cluster hypothesis”.

At the beginning of this section, we have explained that data mining techniques perform pattern extraction. Therefore, they are only capable of discovering alarm patterns, rather than alarm groups. This rose the need to determine the alarm patterns that root causes typically induce, so that one could subsequently devise a data mining technique to extract them. The outcome of this effort is the alarm cluster hypothesis, which can be rephrased as: “For the most part, root causes induce generalized alarms that adequately model large alarm groups”. Even though the meaning of “large” and “adequate” is still vague, it follows from our discussion that a data mining technique capable of extracting this kind of generalized alarms is suitable for root cause analysis. Chapter 4 will give the terms “large” and “adequate” a definite meaning. Moreover, it will propose a data mining technique for extracting the resulting kind of generalized alarms. The remainder of this chapter is dedicated to further validating the alarm cluster hypothesis.

3.3.2 Generalization and Discussion

Given that the alarm cluster hypothesis was derived by means of examples, we now examine its generality. To this end, two separate issues need to be discussed: First, the claim that most root causes manifest themselves in large alarm groups, and second, the claim that these alarm groups can be adequately modeled by generalized alarms.

To get some insight into the size of alarm groups, we turn to the genesis diagram of Figure 3.1. According to the third level of this diagram, root causes can have four different origins:

Intentional & malicious: Many root causes of this origin manifest themselves in large alarm groups. This is particularly true for worms [192] and denial of service attacks [51]. However, reconnaissance scans can also trigger many alarms. Moreover, any attack that needs some tweaking to work properly (e.g. password guessing or buffer-overflow attacks) can trigger a potentially large number of alarms. (Clearly, stealthy attacks trigger only a few alarms. That’s why they are stealthy.)

Intentional & non-malicious: Root causes of this origin are introduced to satisfy some need. As a consequence, they are not likely to remain inactive, but, quite to the contrary, can be expected to cause many alarms. For example, new protocols, topologies, and system management tools are root causes that were introduced because of the value they add to the system users or administrators. Therefore, we expect these root causes to be used actively, and to manifest themselves in large alarm groups.

Inadvertent failures and misconfigurations: From network management, it is known that failures and misconfigurations tend to cause large amounts of alarms [75, 98, 116] (see also Section 2.2.2).

Inadvertent trickery: Root causes of this origin are mostly viruses, which are only marginally detected by IDSs. Therefore, at the time of this writing, root causes of this origin play a minor role for intrusion detection.

From this discussion, it becomes clear that many root causes manifest themselves in relatively large alarm groups. The precise meaning of “large” depends on the time-frame over which a root cause has been active. For example, the longer a worm is active, the more alarms it obviously triggers. Finding a threshold to define “largeness” is an important problem that the next chapter will address. For the time being, “large” can be taken to mean thousands of alarms per month.

We now consider the second claim of the alarm cluster hypothesis, namely that alarm groups can be adequately modeled by generalized alarms. To defend this claim, let us consider an arbitrary root cause and let G be the alarm group, in which this root cause manifests itself. Using the following two-step procedure, it is possible to model G by means of a generalized alarm **g**:

1. Represent G by a single *meta-alarm* $\mathbf{m}$, which is obtained by forming the attribute-wise union of the alarms in G, i.e. set $\mathbf{m}[A_i] := \cup_{\mathbf{a}\in G}\mathbf{a}[A_i]$ for all alarm attributes A_i, where “$\cup$” denotes the union of multi-sets. For example, if $G = {<}(1, A, \beta), (2, B, \alpha), (3, B, \alpha){>}$ is given, then $\mathbf{m} = ({<}1, 2, 3{>}, {<}A, B, B{>}, {<}\alpha, \alpha, \beta{>})$ follows. (Recall that we use pointed brackets, i.e. “$<$” and “$>$”, for multi-sets.)
2. For each attribute A_i, interpret the multi-set $\mathbf{m}[A_i]$ in terms of real-world concepts, and find a descriptive concept name g_i for it. This yields the generalized alarm $\mathbf{g} = (g_1, \ldots, g_n)$. In the above example, we might choose $g_1 = small\text{–}integer$, $g_2 = B$, $g_3 = \alpha$, and hence, $\mathbf{g} = (small\text{–}integer, B, \alpha)$.

Obviously, generalized alarms are capable of modeling arbitrary alarm groups. Note, however, that in the above example, the generalized alarm **g** does not exactly and unambiguously represent the original alarm group G. For example, **g** does not match the alarm $(1, A, \beta)$, even though it is contained in the alarm group G. Conversely, **g** does match the alarm $(1, B, \alpha)$, despite the fact that it is not contained in G. This raises the issue of how adequately generalized alarms model alarm groups.

Intuitively, a generalized alarm adequately models an alarm group if it correctly captures the key information contained in the alarm group. Unfortunately, not all information can be captured by generalized alarms. In particular, the first of the above two steps destroys information about the association between attribute values. For example, given the above meta-alarm $\mathbf{m}$ it is not clear whether or not G contains an alarm in which the attribute values 1 and B co-occur in the same alarm. Our experience has shown that losing all information about the co-occurrence of attribute values can, indeed, negatively influence model adequacy.

The amount of information lost in the second step depends on whether or not there are good concept names to describe the multi-sets $\mathbf{m}[A_i]$. Clearly, the more descriptive the concept names, the smaller the loss of information. Fortunately, we found the information loss due to inadequate concept names to be minor. In general, once root causes had been understood, the concept names for the multi-sets $\mathbf{m}[A_i]$ followed naturally. Nonetheless, step two does destroy information about the frequency of attribute values. For example, the above generalized alarm $\mathbf{g} = (small\text{--}integer, B, \alpha)$ does not show whether the alarm group G contains one, ten, or a hundred alarms with a value of 1 for the first attribute. To compensate for this loss of information, we will later complement generalized alarms with descriptive statistics that show the frequencies of attribute values.

In summary, this section has explained why root causes tend to manifest themselves in large alarm groups, and why in most cases, generalized alarms are an adequate model for these alarm groups. In other words, the alarm cluster hypothesis was shown to be plausible and rationally justifiable.

3.4 Testing the Alarm Cluster Hypothesis

According to the alarm cluster hypothesis, most root causes manifest themselves in large alarm groups that are adequately modeled by generalized alarms. However, the reasoning that lead to this hypothesis was purely theoretical. Therefore, it is advisable to experimentally check if the alarm cluster hypothesis is supported by real-world intrusion detection alarms. The present section describes such an experimental verification of the alarm cluster hypothesis.

At first glance, the experiments of Chapter 6 seem to offer sufficient experimental evidence for the correctness of the alarm cluster hypothesis. However, the experiments of Chapter 6 are not strictly objective because they involve the experimenter's judgment. One could have eliminated this subjectivity by documenting in great detail the setup of the experiments, their results, and the evaluation of these results. This, however, would have added an estimated 100 pages to this thesis, which did not seem practical. The experiments presented next are objective in a formal sense, and they are simple enough to be presented in full detail.

The methodology used in the experiments is analogous to the one used in *tests of cluster tendency* [23, 56, 80, 104]. Such tests decide if a given dataset D can reasonably be assumed to contain clusters, even though the clusters themselves are not identified. In a nutshell, tests of cluster tendency use a *test statistic* that measures in a single number the degree to which a dataset contains clusters. Moreover, they determine the probability that a random dataset scores the same or a better value in the test statistic. If this probability is negligible (say, smaller than 0.001) then the dataset D is assumed to contain clusters. This, however, is no proof for the existence of clusters. In fact, there are limits to how well a single number can measure the existence of clusters. Moreover, even when the dataset D scores a value in the test statistic that is highly unlikely for random datasets, this does not exclude the possibility that D is actually random and rid of any clusters. Thus, tests of cluster tendency are plausibility checks that offer corroborating evidence for the existence of clusters. Analogously, the test presented in this section strengthens our confidence in the alarm cluster hypothesis, but cannot prove it.

In Section 3.4.1, we describe the experiment that we designed to test the alarm cluster hypothesis. Moreover, the results obtained are presented. Section 3.4.2 explains in more detail the mathematics that are used in the experiment.

3.4.1 Experimental Setup and Results

The goal of this section is to experimentally validate the alarm cluster hypothesis. To this end, we first derive a test statistic $\phi_p(\cdot)$ that maps alarm logs to integers. Specifically, for a given alarm log L, the test statistic $\phi_p(L)$ measures how well L supports the alarm cluster hypothesis. The proposed test statistic will return small values to indicate strong support. Unfortunately, there is no obvious threshold below which $\phi_p(L)$ is "small enough" to confirm the alarm cluster hypothesis. Therefore, we proceed in analogy to tests of cluster tendency, and define that $\phi_p(L)$ is "small enough" if a random alarm log has a negligible probability of scoring a ϕ_p-value that is equal to or smaller than $\phi_p(L)$.

We desire the test statistic $\phi_p(L)$ to measure whether the alarm log L contains — as predicted by the alarm cluster hypothesis — large alarm groups that are adequately modeled by generalized alarms. However, as explained in Section 3.2.1, the test statistic $\phi_p(L)$ cannot decide if a multi-set of alarms is an alarm group. Therefore, the best that the test statistic $\phi_p(L)$ can possibly measure is the existence of "large *multi-sets of alarms* that are adequately modeled by generalized alarms". The problem with this formulation is that the terms "large" and "adequately" are too vague to be tested in a formal way.

We begin by making the meaning of "adequate" more precise. Recall that a generalized alarm adequately models a multi-set of alarms if it correctly captures the key features of the multi-set. Therefore, alarms themselves are the "most adequate" generalized alarms because they are maximally specific and do not sacrifice any information to the generalization of attribute values. However, alarms can only model multi-sets of identical alarms. By contrast, the alarms that arise in the real world are generally mutually distinct. Therefore, alarms are too inflexible to model anything but the most trivial multi-sets. More flexibility is needed.

The following observation points the way to a more flexible but still "adequate" class of generalized alarms: For most alarms, moderate modifications of the source port value, the time-stamp value, or the context value do not fundamentally change the alarm's semantic. In fact, the source port value is mostly set at random. Time is specified in units of seconds, even though a granularity of hours or less is generally sufficient. Finally, the context attribute stores raw network packets, which contain a lot of noise in addition to the actual attack. On the other hand side, the source IP address, the destination IP address, the destination port, and the alarm type cannot be modified without substantially changing the meaning of an alarm. This motivates the following working definition:

Definition 3.3 *A generalized alarm is an* **adequate model** *of a multi-set of alarms if it contains an exact (i.e. ungeneralized) value for the source IP address, the destination IP address, the destination port, and the alarm type, while permitting arbitrary values for all other attributes.* □

This definition of adequacy is not the only one possible, but it certainly is a reasonable one. For example, let G be a multi-set of alarms that can be modeled by the generalized alarm $\mathbf{g} \equiv ([Src\text{–}IP = 10.3.2.1] \wedge [Dst\text{–}IP = 10.3.2.2] \wedge [Dst\text{–}port = 80] \wedge [Alarm\text{–}type = 10])$. (Attributes that $\mathbf{g}$ does not specify can assume arbitrary values.) If $\mathbf{g}$ is given, then we know with absolute precision the most important attribute values for the alarms in G. For example, we know that all alarms in G have the source IP address 10.3.2.1 rather than, say, 10.3.2.0, which can make a big difference. It is this specificity with respect to the values of key attributes that makes $\mathbf{g}$ an adequate model for G. For brevity, we call a generalized alarm *adequate* if it is an adequate model for a multi-set of alarms.

To define the test statistic $\phi_p(\cdot)$, let L be an alarm log of size n, i.e. $n = |L|$. Let M be the set of adequate generalized alarms that is obtained by projecting L on the four attributes source IP, destination IP, destination port, and alarm type, which characterize an adequate generalized alarm. Note that each alarm $\mathbf{a} \in L$ matches a $\mathbf{g} \in M$, and conversely, each $\mathbf{g} \in M$ is matched by at least one $\mathbf{a} \in L$. Let $m := |M|$ denote the size of M, let $\varphi(\mathbf{g})$, $\mathbf{g} \in M$, be the number of alarms in L that match $\mathbf{g}$, and let the indices $i_1, \ldots, i_m$ be such that $\varphi(\mathbf{g_{i_1}}) \geq \varphi(\mathbf{g_{i_2}}) \geq$

$\ldots \geq \varphi(\mathbf{g_{i_m}})$. For any fraction $p \in]0, 1]$, the test statistic $\phi_p(L)$ is defined as the smallest integer k for which $\sum_{z=1}^{k} \varphi(\mathbf{g_{i_z}}) \geq \lceil p \times n \rceil$ holds. Intuitively, the test statistic tells you that the $\phi_p(L)$ most frequently matched generalized alarms in M match at least $100 \times p$ percent of the alarms in L.

Suppose that p is large (say, $p = 0.85$) and $\phi_p(L)$ is small in comparison to m. Then, the majority of alarms (namely, at least $\lceil p \times n \rceil$) match one out of a small set of $\phi_p(L)$ generalized alarms. It follows that on the average, each of these generalized alarms must be matched by $\lceil p \times n \rceil / \phi_p(L)$ alarms. Given our assumptions that p is large and $\phi_p(L)$ is small, we conclude that the quotient $\lceil p \times n \rceil / \phi_p(L)$ is large. As a consequence, each of the $\phi_p(L)$ generalized alarms models a *large* multi-set of $\lceil p \times n \rceil / \phi_p(L)$ alarms on the average. For the alarm log L, this implies that it consists of "large multi-sets of alarms that are adequately modeled by generalized alarms". Hence, the alarm log L supports the alarm cluster hypothesis.

This raises the need to decide in a quantitative manner when $\phi_p(L)$ is "small enough" to support the alarm cluster hypothesis. In analogy to tests of cluster tendency, we decide that $\phi_p(L)$ is "small enough" if a random alarm log L' has a probability of at most 0.00001 to score a value $\phi_p(L')$ that is equal to or smaller than $\phi_p(L)$. In other words, $\phi_p(L)$ is "small enough" if the condition $P[\phi_p(L') \leq \phi_p(L) \mid L' \text{ is random}] \leq 0.00001$ holds. The threshold probability 0.00001 is arbitrary, and any other small probability could have been chosen. Random alarm logs must satisfy $|L'| = |L| = n$, and each alarm $\mathbf{a} \in L'$ must have a $\mathbf{g} \in M$, such that $\mathbf{a}$ matches $\mathbf{g}$. These requirements guarantee that the alarm logs L' and L are comparable [23, 104]. Conceptually, random alarm logs are obtained by repeating the following experiment n times: With all generalized alarms in M having the same probability, randomly choose one of them, generate an alarm that matches it, and add this alarm to L'.

By way of illustration, the first IDS of Table 1.2 triggers $n = 42018$ alarms in January 2001, and the set M consists of $m = 7788$ generalized alarms. For $p := 0.85$, we obtain $\phi_p(L) = 2238$. In other words, using 2238 out of the 7788 generalized alarms in M, it is possible to model 85% of the 42018 alarms in L. Moreover, a random alarm log L' satisfies $\phi_p(L') \leq \phi_p(L)$ with a probability of less than 0.00001 (see Section 3.4.2 for a proof). Therefore, the alarm cluster hypothesis is supported by the alarm log that IDS-1 generates in January 2001. This is indicated by a tick in row 1 and column "Jan" of Table 3.2. The other entries of the table can be interpreted in the same way: Each IDS and month defines a separate alarm log, and a tick in the corresponding field indicates that the alarm log supports the alarm cluster hypothesis. A dash, by contrast, stands for no support. In all experiments, p is set to 0.85, and the "IDS" column of Table 3.2 refers back to Table 1.2 on page 7.

Table 3.2: Alarm logs that support the alarm cluster hypothesis (p=0.85).

IDS	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	√	√	√	√	√	√	√	√	√	√	√	√
2	√	√	√	√	√	√	√	√	√	√	√	√
3	√	—	—	—	—	—	—	—	—	—	—	—
4	√	√	√	√	√	√	√	√	√	√	√	√
5	√	√	√	√	√	√	√	√	√	√	√	√
6	√	√	√	—	—-	√	√	√	√	√	√	√
7	√	√	√	√	√	√	√	√	√	√	√	—
8	√	√	√	√	√	—	—	—	√	√	√	√
9	√	√	√	√	—	√	√	√	√	√	—	—
10	√	√	√	√	√	√	√	√	√	√	√	√
11	√	√	√	√	√	√	√	√	√	√	√	√
12	√	√	√	√	√	√	√	√	√	√	√	√
13	√	√	√	√	√	√	√	√	√	√	√	√
14	—	—	—	√	√	—	√	—	√	—	√	—
15	√	√	√	√	√	√	√	√	√	√	√	√
16	√	√	√	√	√	√	√	√	√	√	√	√

It follows from Table 3.2 that 165 out of 192 alarm logs confirm the alarm cluster hypothesis. That offers strong evidence in favor of the alarm cluster hypothesis. In experiments not documented here, we have shown that this result is robust with respect to variations in the definition of adequacy. Moreover, variations in the value of p do not fundamentally change the result, either. However, we have not experimented with other test statistics or other formalizations of the random log concept. Both could affect the results.

3.4.2 Derivation of Probabilities

The last section considered an alarm log L to support the alarm cluster hypothesis if the probability $P[\phi_p(L') \leq \phi_p(L) \mid L'$ is random$]$ is smaller than 0.00001. Here, we address the problem of calculating this probability. However, experience with similar problems [23, 104] suggests that it is very difficult to determine the exact value of the probability $P[\phi_p(L') \leq \phi_p(L) \mid L'$ is random$]$. We will therefore overestimate this probability. Note that an overestimate makes us err at the expense of the alarm cluster hypothesis. In other words, the number of ticks in Table 3.2 could only have increased if the exact probabilities had been used.

Let L, M, $n = |L|$, $m = |M|$, $p \in]0, 1]$, and $\phi_p(\cdot)$ be as in Section 3.4.1, and set $k := \phi_p(L)$. Recall that a random alarm log is defined as the result of iterating the following experiment n times: Randomly choose a generalized alarm from M, generate an alarm that matches this generalized alarm, and add this alarm to the random alarm log under construction. Note that all generalized alarms in M are equally likely to be chosen. These introductory remarks set the stage for the following proposition:

Proposition 3.2 *Let n, m, p, $\phi_p(\cdot)$, and k be as above. For a random alarm log L', and for $\lambda := \lceil p \times n \rceil$, the following inequality holds:*

$$P[\phi_p(L') \leq k] \ \leq \ \frac{1}{m^n} \cdot \binom{m}{k} \cdot \sum_{i=0}^{n-\lambda} \binom{n}{i} k^{n-i}(m-k)^i \tag{3.1}$$

Proof : Attach an imaginary dart board to each generalized alarm in M and imagine throwing n darts at the m dart boards. Suppose that you hit each dart board with the same probability, namely $1/m$. Then, $P[\phi_p(L') \leq k]$ (which is the left-hand side of inequality (3.1)) equals the probability that after throwing all n darts, there are k dart boards that in total have λ or more darts sticking. We will use this more intuitive formulation of the problem, to prove inequality (3.1).

To generate all constellations where k dart boards have at least λ darts sticking, we can proceed as follows: Choose k dart boards and j darts, with $j = \lambda, \ldots, n$. There are $\binom{m}{k} \times \binom{n}{j}$ ways to do this. Moreover, there are $k^j \times (m-k)^{n-j}$ ways to throw the darts so that the selected j darts hit one out of the selected dart boards, whereas the remaining $n-j$ darts hit some non-selected dart board. By summing over all j, we see that this process generates

$$\sum_{j=\lambda}^{n} \binom{m}{k} \binom{n}{j} k^j (m-k)^{n-j} \tag{3.2}$$

constellations. Note, however, that some constellations are generated multiple times. Formula (3.2) counts each constellation as many times as it is generated, and therefore overestimates the actual number of distinct constellations. It is easy to transform formula (3.2) into $\binom{m}{k} \cdot \sum_{i=0}^{n-\lambda} \binom{n}{i} k^{n-i}(m-k)^i$ (just move $\binom{m}{k}$ before the summation, substitute $j := n - i$, and observe that $\binom{n}{n-i} = \binom{n}{i}$). Finally, formula (3.1) is obtained by dividing this quantity by m^n, where m^n is the total number of ways to throw n darts at m dart boards. □

Two notes are in order. First, we researched ways to improve the bound given by Proposition 3.2, but the resulting formulae were complex and did not make a big difference in practice. Second, while calculating Table 3.2, there were nine instances where we found the estimate of formula (3.1) to be too coarse. In these instances, we used a Monte Carlo simulation to obtain a better estimate of the probability.

3.5 Experience with Episode Rules

If one is willing to accept the alarm cluster hypothesis, then clustering stands out as the most adequate data mining technique for root cause analysis. Therefore, the Chapters 4 through 6 focus on clustering as a tool for root cause analysis. However, in network management, researchers have successfully used episode rules in a framework similar to ours [119]. Therefore, it seems natural to mine intrusion detection alarm logs for episode rules. Moreover, given that episode rules and clustering search for very different alarm patterns, a success in the use of episode rules would cast doubts on the appropriateness of clustering. Therefore, we next define episode rules (cf. Section 3.5.1), and then report our experience in using them for root cause analysis (cf. Section 3.5.2).

3.5.1 Definitions

To formally define episode rules, we need the following terminology [142, 143]: An *alarm predicate* is a boolean expression that tests certain alarm attributes such as the alarm type or the source IP address. A *serial (parallel) episode* is a sequence (multi-set) $\alpha =<P_i>_{1\leq i\leq n}$ of alarm predicates. Note that the predicates of a serial episode are ordered, whereas they have no order in parallel episodes. Given a parallel episode α and an alarm sequence S, a time interval $[t_s, t_e]$ is an *occurrence* of α if it contains a *distinct* alarm $\mathbf{a}$ for each P_i such that $P_i(\mathbf{a})$ holds. For occurrences of serial episodes, the alarm order must additionally match the predicate order (i.e. the alarm $\mathbf{a}$ satisfying P_i must occur before the alarm $\mathbf{a}'$ satisfying P_{i+1}). The interval $[t_s, t_e]$ is a *minimal occurrence* of α if there is no proper subinterval of $[t_s, t_e]$ that would also be an occurrence of α. Finally, *episode rules* are implication rules of the form

$$<P_1, \ldots, P_k> \implies <P_1, \ldots, P_k, \ldots, P_n> \ [s, c, W], \tag{3.3}$$

where $<P_i>_{1\leq i\leq k}$ is a sub-episode of $<P_i>_{1\leq i\leq n}$, and the two episodes are either both serial or parallel. The parameters s, c, and W are called *support*, *confidence*, and *window width* and their interpretation is the following: Episode $<P_i>_{1\leq i\leq n}$ has s minimal occurrences in sequence S. Moreover, if $t_e - t_s \leq W$ and $[t_s, t_e]$ is a minimal occurrence of episode $<P_i>_{1\leq i\leq k}$, then there is a c percent probability for the super-episode $<P_i>_{1\leq i\leq n}$ to occur in $[t_s, t_s + W]$. Variations of these definitions are described in [142, 143].

3.5.2 Experience

We have used episode rules to support root cause analysis. In our experiments, we have mined the alarms from our experimental IDSs (cf. Table 1.2) for serial and parallel episodes and episode rules. The set of admissible alarm predicates was restricted to predicates of the form $P(\mathbf{a}) \equiv (\wedge_i \mathbf{a}[A_i] = c_i)$, where $\mathbf{a}$ is an alarm, the A_i are attributes, and the c_i are constants. The episodes and episode rules discovered contained interesting patterns, including the following ones:

- We discovered episodes that were characteristic of attack tools. For example, we repeatedly found episodes that resulted from attack scenarios like the one in Figure 3.3, where a source host triggers the same sequence of alarm types against different target hosts. In general, the root cause of a scenario like this is an attacker who tries out his or her attack tool against different targets.
- We discovered alarms that — for some IDS-specific reason — almost always entail other alarms. For example, on IDSs from one vendor, "TCP FIN Host Sweep" alarms imply "Orphaned FIN Packet" alarms with a confidence of 100% (but not vice versa).
- We discovered episodes that resulted from legitimate system operations such as remote file-system mounts or certain network management tasks.

In general, however, we found that episode rule mining tended to produce a large number of irrelevant or redundant patterns. This is a well-known weakness of episode rules [119]. Moreover, many of the discovered episodes and episode rules were difficult to interpret in terms of root causes. Thus, given a large number of not always easy to interpret episodes and episodes rules, locating the truly interesting ones became a difficult and time-consuming activity. Finally, despite our efforts, we could only identify the root causes of one to five percent of all alarms. Given this low success rate in combination with the high effort required, we concluded that episode rule mining is not particularly suitable for supporting the root cause analysis of intrusion detection alarms.

Digression on attack detection: Our experience with episode rules has shown that intrusion detection alarms are extremely monotonous and repetitive. Specifically, we noticed that almost all high-support episode rules consisted of multiple instances of the same predicate, i.e. $P_i = P_j$ generally held for all i and j in equation (3.3). The monotony of intrusion detection alarms is illustrated more clearly by the following simple experiment: Let us randomly choose an IDS and a source IP address that has triggered alarms at this IDS. This source IP address might have triggered many alarms throughout the year 2001, but with a probability of 96% they were all of the same alarm type! Moreover, the probability for all alarms to

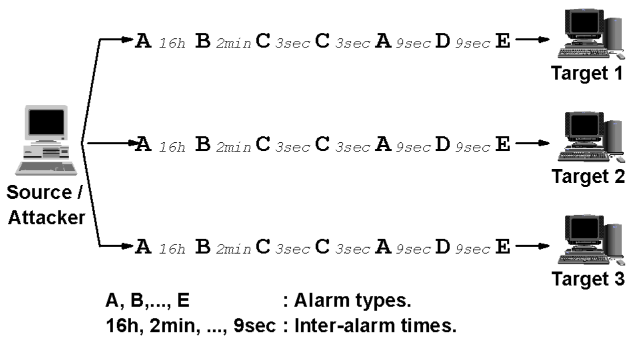


Figure 3.3: An attack tool being run against three targets.

hit the same destination port (destination IP address) is 95% (69%). These probabilities were calculated using the 16 IDSs in Table 1.2, but we have confirmed them (give or take a few percentage points) using over 90 million alarms from more than 50 different IDSs.

The above observation inspires a simple paradigm for detecting attackers: Source IP addresses that display diverse behavior, e.g. by triggering alarms of many different types, are likely to perform an attack. In fact, hackers generally have little a priori knowledge about their targets and therefore resort to trying different reconnaissance and attack techniques until they are successful or exhausted. In doing so, hackers tend to trigger diverse alarm streams that involve many different alarm types and targets. Given that this kind of diverse behavior is generally rare, it is a rewarding heuristic to investigate it more closely when it occurs. Note, however, that perfectly monotonous behavior (e.g. password guessing) can still constitute an attack. Therefore, zooming in on diverse behavior helps in finding real attacks, but not all attacks are diverse.

Chapter 4

Alarm Clustering

The present chapter describes the alarm clustering method that we have developed for root cause analysis. To explain the need for a new clustering method, we begin in Section 4.1 with a brief review of the field of cluster analysis. Section 4.2 uses the alarm cluster hypothesis (cf. Proposition 3.1) as a starting point to derive an algorithmic framework for alarm clustering. The implementation of this framework is described in Section 4.3. Possible extensions of the framework are discussed in Section 4.4.

4.1 Introduction to Cluster Analysis

The purpose of this section is to survey the field of cluster analysis, and to explain why we found it necessary to derive the new clustering method described in the Sections 4.2 and 4.3. The reader not interested in this background material can skip to Section 4.2 without loss of continuity.

Clustering seeks to group objects into categories (called *clusters*) so that the objects within a given category are alike, while they are different from objects in other categories [9, 12, 81, 104]. The degree of alikeness between a pair of objects is measured by a proximity index. We summarize the most widely used proximity indices in Section 4.1.1. Section 4.1.2 gives an overview of several popular clustering methods. This overview is necessarily incomplete because the number of known clustering methods ranges in the hundreds [11]. Unfortunately, virtually all of these methods are largely unexplored with respect to their strengths and limitations. This makes it a challenging task to choose a suitable clustering method for a given applied analysis. Section 4.1.3 discusses this problem and explains the need for the new clustering method that is presented in the remaining sections of this chapter.

Our notation is as follows: Let D denote the dataset to be clustered, and let n be its size (i.e. $n = |D|$). The elements in D are referred to as *objects*. In cluster analysis, it is common to represent the objects $\mathbf{x_i} \in D$ as p-dimensional tuples, i.e. $\mathbf{x_i} = (x_{i1}, \ldots, x_{ip})$, $i = 1, \ldots, n$. The tuple dimensions are called *attributes*, and the x_{ik}s are known as *attribute values* or *measurements*. The dataset D is called *numerical*, *binary*, or *categorical*, if all attributes are numerical, binary, or categorical, respectively. The dataset D is *mixed* if it contains attributes of at least two different types (e.g. numerical and binary).

4.1.1 Proximity Indices

Proximity indices measure the degree of alikeness between a pair of objects. There are two types of proximity indices, namely similarity indices and distance indices. *Similarity indices*, such as correlation, return a large value to indicate a high degree of alikeness, whereas *distance indices* (e.g. Euclidean distance) return a small value for this case. This subsection describes the most commonly used proximity indices. A broader and more detailed treatment of proximity indices can be found in [9, 81, 90, 104].

For numerical datasets, the most widely used distance index is the Minkowski metric. Specifically, for a fixed $r \geq 1$, the Minkowski distance $d_r(\mathbf{x_i}, \mathbf{x_j})$ between the objects $\mathbf{x_i} = (x_{i1}, \ldots, x_{ip})$ and $\mathbf{x_j} = (x_{j1}, \ldots, x_{jp})$ is defined as:

$$d_r(\mathbf{x_i}, \mathbf{x_j}) := \left(\sum_{k=1}^{p} |x_{ik} - x_{jk}|^r \right)^{(1/r)} \tag{4.1}$$

The two most commonly used Minkowski metrics are the Euclidean distance and the Manhattan distance, which are obtained for $r = 2$, and $r = 1$, respectively. The squared Mahalanobis distance [5, 104] and Pearson's correlation coefficient [5, 81] are two further proximity indices for numerical datasets.

For binary datasets, the similarity between two objects $\mathbf{x_i} = (x_{i1}, \ldots, x_{ip})$ and $\mathbf{x_j} = (x_{j1}, \ldots, x_{jp})$, $x_{ik}, x_{jk} \in \{0, 1\}$, is most conveniently expressed in terms of four counts a_{00}, a_{01}, a_{10}, and a_{11}, where a_{uv} equals the number of indices $k \in \{1, \ldots, p\}$, for which $x_{ik} = u$ and $x_{jk} = v$ holds. Note that the four a_{uv} values sum up to p, the total number of attributes. Several similarity indices can be defined from the four numbers a_{00}, a_{01}, a_{10}, and a_{11}. Anderberg reviews most of them and puts them into context [9]. Two particularly common similarity indices are the simple matching coefficient $s_M(\mathbf{x_i}, \mathbf{x_j}) := (a_{00}+a_{11})/(a_{00}+a_{01}+a_{10}+a_{11})$ and the Jaccard coefficient $s_J(\mathbf{x_i}, \mathbf{x_j}) := a_{11}/(a_{01}+a_{10}+a_{11})$. Note that the simple matching coefficient weights matches of 0's the same as matches of 1's, whereas the Jaccard coefficient ignores matches of 0's.

Categorical datasets generalize binary ones, in that categorical attributes can assume more than two values. Accordingly, many proximity indices for the categorical case are straightforward generalizations of their binary counterparts [9, 90, 100]. For example, given two objects $\mathbf{x_i}$ and $\mathbf{x_j}$, the simple matching coefficient has been redefined as $s_M(\mathbf{x_i}, \mathbf{x_j}) := m/p$, where m is the number of attributes on which the two objects agree, i.e. $m := |\{k \,|\, x_{ik} = x_{jk}\}|$. Similarly, it has been suggested to encode categorical datasets by binary ones, so one can directly apply the proximity indices from the binary case [9, 90].

So far, it has been assumed that all attributes of the dataset D have the same type, where potential types are either numerical, binary, or categorical. In practice, mixed datasets having attributes of at least two different types are very common. Then, the distance $d(\mathbf{x_i}, \mathbf{x_j})$ between two objects $\mathbf{x_i} = (x_{i1}, \ldots, x_{ip})$ and $\mathbf{x_j} = (x_{j1}, \ldots, x_{jp})$ is generally calculated as the sum of normalized attribute-distances [9, 117]. Hence, $d(\mathbf{x_i}, \mathbf{x_j}) := \sum_{k=1}^{p} \delta(x_{ik}, x_{jk})$, where $\delta(x_{ik}, x_{jk}) \in [0, 1]$ is the normalized distance between the measurements x_{ik} and x_{jk}. Alternatively, a proximity index can be defined by explicitly enumerating the proximities p_{ij} between all pairs $(\mathbf{x_i}, \mathbf{x_j})$ of objects. In this case, the proximity index is given by the $n \times n$ *proximity matrix* $(p_{ij})_{1 \le i,j \le n}$, whose rows and columns correspond to the objects in D, and whose entries p_{ij} specify the proximities between the objects $\mathbf{x_i}$ and $\mathbf{x_j}$.

4.1.2 Overview of Clustering Methods

Clustering methods organize a given dataset into an “adequate” number of clusters. Clustering methods have been classified according to the kind of processing they perform [90, 185, 203]. This gives rise to five classes, namely *partitioning methods*, *hierarchical methods*, *density-based methods*, *grid-based methods*, and *model-based methods*. This classification is neither mutually exclusive nor exhaustive [62, 104], but given its popularity in data mining, we will still use it to organize the following discussion.

Partitioning methods take two inputs: First, a dataset $D = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ of n objects, and second, an integer K, indicating the desired number of clusters. Then, a partitioning method splits the given dataset into K non-overlapping clusters $\{C_1, \ldots, C_K\}$, so that a criterion of cluster quality is optimized. A *criterion of cluster quality* is a mathematical formula that captures one's intuition that “good” clusters should be compact and isolated. Assuming that the objects $\mathbf{x_i} \in D$ are drawn from an Euclidean space, the *squared-error* E_K^2 is by far the most commonly used criterion of cluster quality:

$$E_K^2(C_1, \ldots, C_K) := \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} [d_2(\mathbf{x}, \mathbf{m_k})]^2, \tag{4.2}$$

where $d_2(\cdot,\cdot)$ denotes Euclidean distance, and $\mathbf{m_k} := (1/|C_k|) \times \sum_{\mathbf{x} \in C_k} \mathbf{x}$ is the mean of cluster C_k. Aside from squared-error, many other criteria of cluster quality have been suggested [81, 91]. However, squared-error makes good intuitive sense and is efficient to compute.

The problem of partitioning a given set of n objects into K clusters so as to optimize a stated quality criterion is, in general, NP-complete [45]. This fact precludes an exhaustive search and has encouraged the development of simple heuristic algorithms [9, 104]. The most popular of these is the K*-means* method, which locally minimizes the squared-error criterion E_K^2. The K-means method (cf. Figure 4.1) starts with a randomly chosen set of cluster means (step 1) and then repeatedly relocates objects so as to decrease the squared-error E_K^2 (steps 2 to 7). Object relocation continues until the clusters no longer change between consecutive iterations. The K-means method has been proven to terminate [184], but the resulting partition is frequently a local rather than a global optimum [5, 184]. Moreover, the cluster means initially selected in step 1 of Figure 4.1 are known to significantly impact the quality of the results [5, 25, 148, 150, 151].

Variations of the K-means methods have mainly focused on three aspects: First, the strategy used to select the initial cluster means [9, 25], second, the order in which objects are relocated and means are recomputed [9, 140], and third, heuristic ways of automatically adjusting the number K of clusters [9, 15]. Moreover, genetic algorithms, simulated annealing, and other search methods have been used to solve the optimization problem posed by partitional clustering [105]. Note that the K-means method is limited to numerical datasets since it requires the ability to compute means. Huang extends the K-means paradigm to categorical datasets [100]. PAM [117], CLARA [117], and CLARENS [162] are three variants of the K-means method that operate on mixed datasets, provided an appropriate proximity index (e.g. in the form of a proximity matrix) is specified. Moreover, these methods are more robust than K-means in the presence of noise and outliers.

Hierarchical methods transform a dataset into a hierarchically structured sequence of partitions. The root of this hierarchy is the set of all objects, the leaves are the individual objects, and intermediate layers represent partitions of varying granularities.

The most widely used class of hierarchical methods starts with the trivial partition, in which each object is placed into a separate cluster. Each subsequent step takes two clusters of the preceding step and merges them into a single one. This yields a new partition with one cluster less. The merging continues until all objects have been amalgamated into one big cluster. Clearly, the algorithm terminates after n steps, where n is the number of objects in the dataset. The

Input: A dataset $D = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$, and an integer K;
Output: A partition of the dataset D into K clusters $C_1, \ldots, C_K$;
Algorithm:

for $k := 1$ ***to*** K ***do*** Initialize $\mathbf{m_k}$ to be a randomly chosen point from D;
while the clusters C_k change ***do*** {
    for $k := 1$ ***to*** K ***do*** *// Relocate objects:*
        $C_k := \{\mathbf{x} \in D \mid d_2(\mathbf{x}, \mathbf{m_k}) \leq d_2(\mathbf{x}, \mathbf{m_j}),\ \forall j \neq k\}$;
    for $k := 1$ ***to*** K ***do*** *// Compute the new cluster means:*
        $\mathbf{m_k} := (1/|C_k|) \times \sum_{\mathbf{x} \in C_k} \mathbf{x}$;
}

Figure 4.1: The K-means algorithm.

resulting sequence of n hierarchically nested partitions can be represented by a dendrogram, such as the one in Figure 4.2(a). A *dendrogram* is a binary tree whose nodes have an associated height. Specifically, the node at height i shows the two clusters that were merged by step i. Moreover, cutting the dendrogram just above height i yields $1 + n - i$ subtrees, each of which represents one cluster in the partition created by step i. For example, step 5 of Figure 4.2(a) creates the partition $\{\ \{1, 2, 3\}, \{4, 5\}, \{6, 7\}\ \}$, which is shown in Figure 4.2(b).

Let us consider the merge step in more detail. In general, the merge step selects two clusters C_r and C_s whose distance $\Delta(C_r, C_s)$ is minimum among any two clusters, i.e. C_r and C_s must satisfy $\Delta(C_r, C_s) = \min_{i,j}\{\ \Delta(C_i, C_j)\ \}$. Depending on the distance measure $\Delta(\cdot, \cdot)$, one obtains different hierarchical methods. Specifically, let n be the number of objects in the dataset, and let $d(\mathbf{x}, \mathbf{y})$ denote the distance between the objects $\mathbf{x}$ and $\mathbf{y}$. Three widely used measures of inter-clusters distance are:

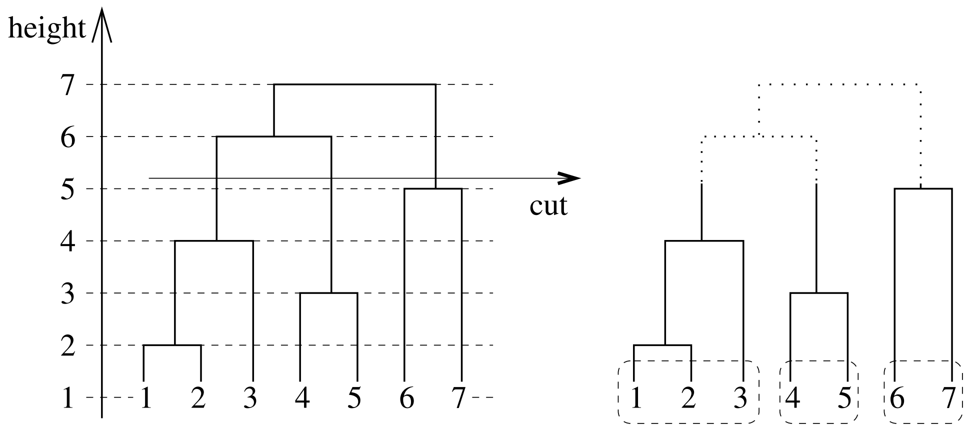
$$\Delta_{min}(C_r, C_s) \quad := \quad \min\{\ d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_r, \mathbf{y} \in C_s\ \} \tag{4.3}$$

$$\Delta_{max}(C_r, C_s) \quad := \quad \max\{\ d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_r, \mathbf{y} \in C_s\ \} \tag{4.4}$$

$$\Delta_{avg}(C_r, C_s) \quad := \quad \frac{1}{|C_r||C_s|} \times \sum_{\mathbf{x} \in C_r} \sum_{\mathbf{y} \in C_s} d(\mathbf{x}, \mathbf{y}) \tag{4.5}$$

Note that the distance $\Delta(\cdot, \cdot)$ between two clusters is a function of the distances $d(\cdot, \cdot)$ between individual objects. Equation (4.3) defines the well-known *single-link method*, which continuously merges the two clusters containing the closest pair of objects, one from each cluster. Equation (4.4) defines the *complete-link method*, and (4.5) defines the *group average method*. Many other definitions for $\Delta(\cdot, \cdot)$ have been suggested [78].

The above class of hierarchical methods is known as *SAHN* (Sequential, Agglomerative, Hierarchical, Non-overlapping) methods. The general time com-



a) A sample dendrogram. b) The partition created by step 5.

Figure 4.2: A sample dendrogram and a partition it encodes.

plexity of SAHN methods is $O(n^2)$, even though algorithms with a smaller time complexity exist for special cases [45]. BIRCH [205] and CURE [83] are two methods that achieve linear run-times in the number of objects, and moreover have better noise-tolerance than SAHN methods. However, both methods rely on vector operations, which restricts their applicability to numerical datasets. CHAMELEON [115] is another hierarchical method with good noise tolerance. ROCK [84], finally, is a hierarchical method that was particularly designed for categorical datasets.

Density-based methods define clusters as dense regions (i.e. regions containing "many" objects per data volume) that are separated by low-density regions. We describe DBSCAN [61] as a representative of the class of density-based methods. For a dataset D and an associated distance index $d(\cdot,\cdot)$, the basic ideas of DBSCAN can be summarized as follows:

1. Given the user-defined parameters ε and *MinObjs*, an object $\mathbf{x} \in D$ is called a *core object* if there are at least *MinObjs* objects in its ε-neighborhood, i.e. $|\{\mathbf{y} \in D \mid d(\mathbf{x},\mathbf{y}) \leq \varepsilon\}| \geq MinObjs$ must hold.
2. An object $\mathbf{y} \in D$ is *density reachable* from an object $\mathbf{x}$ if and only if there is a chain of objects $\mathbf{z_1},\ldots,\mathbf{z_k}$ with $\mathbf{z_1} = \mathbf{x}$ and $\mathbf{z_k} = \mathbf{y}$, such that $\mathbf{z_1},\ldots,\mathbf{z_{k-1}}$ are core objects and $d(\mathbf{z_i},\mathbf{z_{i+1}}) \leq \varepsilon$ for $i = 1,\ldots,k-1$.
3. A *cluster* C is a non-empty subset of D that is *maximal* (i.e. $\forall \mathbf{x} \in C$: if $\mathbf{y} \in D$ is density reachable from $\mathbf{x}$, then $\mathbf{y} \in C$) and *connected* (i.e. $\forall \mathbf{x},\mathbf{y} \in C$: there is an $\mathbf{z} \in C$ such that $\mathbf{x}$ and $\mathbf{y}$ are density reachable from $\mathbf{z}$).

Using the above definition of clusters, DBSCAN finds all clusters in a dataset. DENCLUE [95] generalizes DBSCAN and improves its run-time performance by a factor of up to 45. CACTUS [74] is a density-based method specifically designed for categorical datasets.

Grid-based methods quantize the data space into a finite number of cells that form a grid structure. For example, quantization may break the 3-dimensional cube $[0,3] \times [0,3] \times [0,3] \subset \mathbb{R}^3$ into, say, 27 cells, namely $[i, i+1] \times [j, j+1] \times [k, k+1]$, with $i, j, k \in \{0, 1, 2\}$. Each cell of the resulting grid structure is used to store various statistics about the data objects it contains. The actual clustering operations are performed on the grid structure (rather than on the original objects), which generally results in very fast processing times. WaveCluster [185] and CLIQUE [1] are representative grid-based methods.

Model-based methods can be further subdivided into conceptual clustering methods and mixture models. *Conceptual clustering methods* were motivated by the fact that most conventional clustering methods do not adequately address the problem of representing clusters in an intelligible manner that supports understanding and decision making [81, 105, 146]. Conceptual clustering, by contrast, puts cluster representation in the foreground and searches for clusters that have "good" representations in a given description language [67, 146, 172]. Examples of description languages include variants of predicate logic [21, 88, 146] as well as probabilistic languages that lists the probabilities of attribute values [67, 195]. In addition, conceptual clustering methods are particularly strong at handling categorical datasets. CLUSTER/2 [146], COBWEB [67], and Attribute-Oriented Induction (AOI) [88] are three well-known conceptual clustering methods.

Mixture models, the second type of model-based methods, assume that the dataset D was generated by K multivariate distributions $M_1, \ldots, M_K$. Clustering in this context means to group the objects according to the distributions that generated them. More formally, we want to determine the membership function $\gamma : D \to \{1, \ldots, K\}$ that assigns each object $\mathbf{x} \in D$ to the index of the cluster/distribution $M_{\gamma(\mathbf{x})}$ that generated it. To this end, let $f_k(\mathbf{x}|\theta_k)$, $k = 1, \ldots, K$, be the density function of the k-th distribution M_k, with θ_k being unknown model parameters. Given these definitions, and assuming that the dataset D is independent and identically distributed, mixture models aspire to find the $\gamma(\cdot)$ and $\theta_1, \ldots, \theta_K$ that maximize the probability

$$\mathcal{L}(D) = \prod_{\mathbf{x} \in D} f_{\gamma(\mathbf{x})}(\mathbf{x}|\theta_{\gamma(\mathbf{x})}) \tag{4.6}$$

of observing the dataset D. This optimization problem is typically solved by means of the Expectation-Maximization (EM) algorithm [49]. Mixture models using Gaussian density functions are described in [16]. Mixture-models with a Bayesian twist have been used by the AUTOCLASS system [34, 93]. Finally, note that mixture models must not be used unless the parametric density functions $f_k(\mathbf{x}|\theta_k)$ can be shown to adequately model the data generation process [91].

Other clustering methods that do not fit well in the above classification include overlapping methods [10, 50, 108, 186], fuzzy methods [97, 182], and neural networks [159]. We do not consider these methods any further because their limited scalability or restriction to numerical datasets [81, 105] conflict with our requirements of Section 3.2.2.

4.1.3 Choosing a Clustering Methods

In this section, we consider the problem of choosing a clustering method that is suitable for analyzing a given dataset D. This is a challenging task because there are hundreds of methods to choose from [11], their theoretical properties are largely unexplored [5, 104, 148], and experimental studies have shown that the performance of most clustering methods is strongly affected by the characteristics of the datasets at hand [5, 148, 150].

A pragmatic approach to selecting a clustering method is to choose the method that most closely satisfies the requirements of the application domain [69, 81]. For example, if the dataset D is suspected to be large and noisy, then a scalable and noise-tolerant clustering method is advisable. Similarly, if the application domain requires deterministic results, then one should avoid clustering methods that use sampling. Motivated by these examples, we have identified six features that affect the suitability of clustering methods for a particular dataset D:

Supported data types: The ability of a clustering method to handle numerical, categorical, or mixed datasets strongly influences its applicability to a given dataset. For example, methods such as K-means that require numerical datasets are useless for mixed or categorical datasets.

Time complexity: The time complexity of a method determines its scalability. Clearly, methods with poor scalability are unsuitable for large datasets.

Noise tolerance: Noise (a.k.a. outliers) are objects that do not cluster naturally with any other object [61]. Noisy datasets should only be analyzed by noise-tolerant methods.

Number of input parameters: Many clustering methods are sensitive to their input parameter settings, so that slightly different parameter settings can produce markedly different clustering results [9, 90, 105, 203]. Moreover, there are few guidelines for choosing "good" parameter values. As a consequence, the usability of a clustering method decreases with the number of input parameters [90, 203].

Input order dependence: Some clustering methods yield different results, depending on the ordering of the objects in the input dataset D. In general, input order dependence is undesirable [90].

Random sampling: Some clustering methods use random sampling to improve their scalability [90, 117]. These methods are no longer deterministic, and their random decisions can strongly influence their results [5, 90, 206].

Table 4.1 on the pages 62 and 63 uses these features to characterize the clustering methods of Section 4.1.2. Even though this table is not exhaustive, it is highly representative of the field as described by the widely read monographs [9, 12, 81, 90, 104, 117]. Clearly, a survey table like this is necessarily a simplification. For example, the table does not show that DBSCAN can actually handle mixed datasets, provided quadratic (i.e. $O(n^2)$) run-times are accepted [90]. Similarly, the trichotomy low / medium / high for noise tolerance is clearly simplistic. Furthermore, some values of the table were difficult to determine. For example, the authors of ROCK argue in favor of its noise tolerance [84]. However, Zaiane et al. show that ROCK performs poorly in the presence of noise [203]. We found the arguments by Zaiane et al. more convincing and gave ROCK a ranking of "low". This decision is certainly debatable. Finally, for clustering methods with optional parameters, the table only indicates the minimum number of parameters that must be set. For example, a user of BIRCH *must* set at least four parameters, but *might* choose to tweak any of a number of additional parameters.

Table 4.1 can now be used to choose a suitable clustering method. As explained in Section 3.2.2, our application domain demands a scalable, noise-tolerant, easy-to-use clustering method that supports mixed datasets. Additionally, we require that the method has to be "robust", i.e. deterministic and insensitive to the input order. When applied to Table 4.1, these requirements attach a negative mark to a method if *(1)* it scales quadratically or worse in the number of objects, *(2)* if it has low noise tolerance, *(3)* if it burdens the user with four or more input parameters, *(4)* if it cannot handle mixed datasets, *(5)* if it uses random sampling, *or (6)* if it is input order dependent. For example, the K-means method receives two negative marks, one because it cannot handle mixed datasets, and one because it is sensitive to noise. K-modes also receives two negative marks, one for

Table 4.1: Main characteristics of clustering methods.

Name	Supported data types	Time complexity*	Noise tolerance	No. of input parameters	Input order dependence	Random sampling	Notes
Partitioning methods							
K-means [140]	numerical	$O(kn)$	low	1	no	none	Needs the no. k of clusters as input parameter; initially chosen means strongly affect the results [25]; only suitable when all clusters have similar sizes [90];
K-modes [100]	categorical	$O(kn)$	medium	1	yes	none	
PAM [117]	mixed	$O(k(n{-}k)^2)$	medium	1	no	none	
CLARA [117]	mixed	$O(k^3 + kn)$	medium	2	no	yes	
CLARANS [162]	mixed	$O(k(n{-}k)^2)$	medium	3	no	yes	
Hierarchical methods							
SAHN [81, 104]	mixed	$O(n^2)$	medium	0	no	none	
BIRCH [205]	numerical	$O(n)$	high	≥ 4	yes	none	
CURE [83]	numerical	$O(S^2 + n)$	high	5	no	yes	
ROCK [84]	categorical	$O(S^2 \log S)$	low [203]	2	no	yes	
CHAMELEON [115]	mixed	$O(n^2)$	high	≥ 3	no	none	

* Were n is the number of objects in the input dataset, k is the number of returned clusters, and S is the sample size.

Table 4.1: Main characteristics of clustering methods *(continued)*.

Name	**Supported data types**	**Time complexity***	**Noise tolerance**	**No. of input parameters**	**Input order dependence**	**Random sampling**	**Notes**
Density-based methods							
DBSCAN [61]	numerical	$O(n \log n)$	high	2	no	none	
DENCLUE [95]	numerical	$O(n \log n)$	high	2	no	none	In practice, up to 45 times faster than DBSCAN [95];
CACTUS [74]	categorical	$O(n)$	medium	2	no	none	Tends to produce a large number of clusters [73];
Grid-based methods							
WaveCluster [185]	numerical	$O(n)$	high	≥ 2	no	none	Only suitable for low-dimensional data [185];
CLIQUE [1]	mixed	$O(n)$	medium	2	no	none	Generates readable cluster descriptions [1];
Model-based methods							
COBWEB [67]	categorical	$O(n \log k)$	low [68]	0	yes	none	Time & memory demands tend to be high [90];
AOI [86, 87, 88]	mixed	$O(n \log k)$	medium	≥ 2	no	none	Generates readable cluster descriptions;
Mixture models [16, 34, 71]	*mixed*, if attributes are statistically independent	$O(kn)$	medium	≥ 2	no	none	Local maxima [91], high run-times [26, 70, 71], and model-selection [91] pose problems in practice;

* Were n is the number of objects in the input dataset, k is the number of returned clusters, and S is the sample size.

not supporting mixed datasets, and one for being sensitive to the input order. In fact, CLIQUE, AOI, and mixture models are the only three methods in Table 4.1 that receive no negative marks. Therefore, they are clear candidates for the alarm clustering task.

A closer investigation of CLIQUE reveals that its support for mixed datasets is rather limited [1]. Moreover, Section 4.3.3 will show that AOI is still too noise sensitive for intrusion detection alarms. Finally, the notes column of Table 4.1 indicates some drawbacks of mixture models (see also page 59): First, mixture models can only find local optima. Second, even though their theoretical time complexity is attractive, they do not scale well in practice. Third, it is rather difficult to find parametric density functions that adequately model the alarm generation process. Thus, neither CLIQUE, nor AOI, or mixture models are by themselves fully satisfactory. This justifies the new clustering method of the Sections 4.2 and 4.3. It is interesting to note that this method is a clone between CLIQUE and AOI, with stronger resemblance to AOI.

4.2 A Framework for Alarm Clustering

Chapter 3 has explored the problem of finding a data mining technique that is suitable for root cause analysis. To summarize, an *alarm group* was defined as a multi-set of alarms that share the same root cause. We explained that ideally, a data mining technique should recover alarm groups, but this was shown to be impossible. Therefore, the problem of approximating alarm groups was raised. This lead us to analyze the kind of alarm patterns that root causes typically induce. The idea was that if we understood the manifestations of root causes (i.e. their alarm patterns), then we could discover root causes by spotting their manifestations. This kind of reasoning, called abduction (cf. Section 2.2.2), is not mathematically correct, but very common in practice. Our investigation of root causes and their alarm patterns culminated in the alarm cluster hypothesis (cf. Proposition 3.1) according to which most root causes trigger many alarms, and the multi-sets of these alarms can be adequately modeled by generalized alarms. Therefore, the data mining problem to be solved is one of finding generalized alarms that adequately model large multi-sets of alarms.

From a clustering point of view, this problem becomes one of defining a suitable proximity index. This proximity index must take two alarms as input and return a numerical measure of how adequately these alarms can be modeled by a single generalized alarm. In other words, the proximity index must quantify our intuitive notion of “adequacy”. Section 4.2.1 proposes such an index, which returns small values when an adequate model exists. Hence, the index is a distance index. Section 4.2.2 extends this distance index from pairs of alarms to multi-sets of alarms. Moreover, it defines the alarm clustering problem according to which we want to find multi-sets of alarms that minimize the value of the distance index while having a user-defined minimum size. This minimum size parameter formalizes our hitherto intuitive concept of "largeness". Section 4.2.3 discusses the generality of our alarm clustering framework.

Note that this section proposes a framework for alarm clustering, rather than a single way of clustering alarms. In particular, instead of defining a single distance index, we introduce a principled way of defining such indices. Different instantiations of this framework are possible. For example, the sense of adequacy used in Section 3.4 is one possible instantiation of the framework.

4.2.1 A Distance Index that Measures Adequacy

Section 4.1.1 has shown that few proximity indices exist for categorical attributes. Intrusion detection alarms contain time, numerical, and free-text attributes in addition to categorical attributes (cf. Section 1.4), which further complicates matters. The distance index presented here has its origin in the field of information retrieval, where it has been proposed to use generalization hierarchies to define distance [135, 178, 179, 180]. Very briefly, a generalization hierarchy (a.k.a. *is-a* hierarchy) is a single-rooted directed acyclic graph that shows how concepts are organized into more general concepts. For example, a generalization hierarchy might state that host *mickey* is a Web server, and that host *donald* is a Web server, as well. In this case, we define *mickey* and *donald* to have distance two because the shortest path connecting them via a common ancestor has length two.

Towards a formal treatment of distance, recall that we model alarms as tuples over the n-dimensional attribute space $\text{dom}(A_1) \times \ldots \times \text{dom}(A_n)$, where $\text{dom}(A_i)$ is the domain (i.e. range of possible values) of attribute A_i. A *generalized attribute value* is a concept name that represents a subset of the values in $\text{dom}(A_i)$, $i \in \{1, \ldots, n\}$. For example, the generalized attribute value *Web-server* might represent the set $\{mickey, donald\}$ of two individual hosts. The *extended domain* $\text{Dom}(A_i)$ of attribute A_i is the union of the domain $\text{dom}(A_i)$ and the set of generalized attribute values that have been defined for A_i. (The problem of defining meaningful generalized attribute values is considered in Section 4.2.3.) Finally, given these definitions, a *generalized alarm* is a tuple in $[\text{Dom}(A_1) \times \ldots \times \text{Dom}(A_n)] \smallsetminus [\text{dom}(A_1) \times \ldots \times \text{dom}(A_n)]$.

For each attribute A_i, let $\mathcal{G}_i$ be a single-rooted directed acyclic graph (DAG) on the elements of the extended domain $\text{Dom}(A_i)$. The graph $\mathcal{G}_i$ is called a *generalization hierarchy*. For two elements $x, \hat{x} \in \text{Dom}(A_i)$, we call $\hat{x}$ a *parent* of

x if the generalization hierarchy $\mathcal{G}_i$ contains an edge from $\hat{x}$ to x. The element $\hat{x}$ is an *ancestor* of x if $\hat{x} \neq x$ holds, and there is a directed path from $\hat{x}$ to x (in symbols: $x \lhd \hat{x}$). For $x = \hat{x}$ or $x \lhd \hat{x}$, we write $x \unlhd \hat{x}$, and call $\hat{x}$ a *weak ancestor* of x. The distance $d(x_1, x_2)$ between any two elements $x_1, x_2 \in \text{Dom}(A_i)$ is the length of the shortest path that connects x_1 and x_2 via a common weak ancestor $\hat{x}$, i.e. $d(x_1, x_2) := \min\{\, \delta(x_1, \hat{x}) + \delta(x_2, \hat{x}) \mid \hat{x} \in \text{Dom}(A_i),\ x_1 \unlhd \hat{x},\ x_2 \unlhd \hat{x}\,\}$, where $\delta(\cdot,\cdot)$ measures the length of the shortest path between two nodes[1].

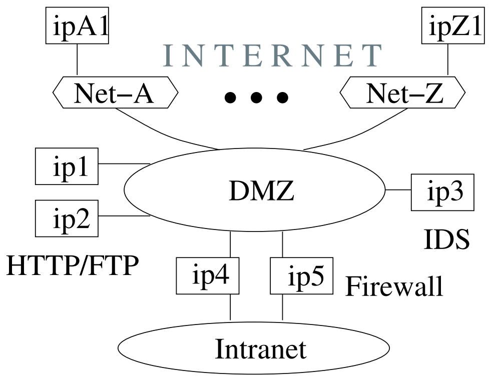
Next, we extend these definitions from attributes to alarms. To this end, let $\mathbf{a}, \hat{\mathbf{a}} \in \mathsf{X}_{1 \le i \le n}\text{Dom}(A_i)$ denote two (possibly generalized) alarms. The alarms $\mathbf{a}$ and $\hat{\mathbf{a}}$ satisfy $\mathbf{a} \unlhd \hat{\mathbf{a}}$ if and only if $\mathbf{a}[A_i] \unlhd \hat{\mathbf{a}}[A_i]$ holds for all attributes A_i. If, additionally, $\mathbf{a}[A_j] \lhd \hat{\mathbf{a}}[A_j]$ holds for at least one attribute A_j, then we write $\mathbf{a} \lhd \hat{\mathbf{a}}$, and say that $\mathbf{a}$ is more *specific* than $\hat{\mathbf{a}}$, while the alarm $\hat{\mathbf{a}}$ is called more *abstract* or *general* than $\mathbf{a}$. The distance $d(\mathbf{a_1}, \mathbf{a_2})$ between any two alarms $\mathbf{a_1}, \mathbf{a_2} \in \mathsf{X}_{1 \le i \le n}\text{Dom}(A_i)$ is defined as the weighted sum of attribute distances, i.e. $d(\mathbf{a_1}, \mathbf{a_2}) := \sum_{i=1}^{n} w_i \times d(\mathbf{a_1}[A_i], \mathbf{a_2}[A_i])$, with weights $w_i \in \mathbb{R}^+$.

By way of illustration, Figure 4.3 shows a network topology, a sample alarm log, and the generalization hierarchies that one might want to use for IP addresses and port numbers in this environment. (For the time being, please ignore the "Count" column in Figure 4.3(b).) Let us verify that the following holds:

- The extended domain Dom(IP) of IP addresses is the union of elementary IP addresses (i.e. the set dom(IP) $= \{p.q.r.s \mid p, q, r, s \in \{0, \ldots, 255\}\,\}$) and generalized IP addresses (i.e. the set {*HTTP/FTP*, *Firewall*, *Net-A*, …, *Net-Z*, *DMZ*, *Internet*, *Any-IP*}). Similarly, the extended domain of port numbers is $\{1, \ldots, 65535,$ *Privileged*, *Non-privileged*, *Any-port*$\}$.
- According to Figure 4.3(c), the IP address $ip1$ is a HTTP and FTP server, which is a DMZ machine, which, more generally, is any IP address. More succinctly, this relationship can be expressed as $ip1 \lhd$ *HTTP/FTP* $\lhd$ *DMZ* $\lhd$ *Any-IP*. Moreover, $ip1 \unlhd ip1$ and $d(ip1, \textit{Any-IP}) = 3$ hold.
- Assuming that all weights equal one, i.e. $w_1 = w_2 = 1$, we find that $d(\,(ip1, ip4),\ (ip1, ipA1)\,) = d(ip1, ip1) + d(ip4, ipA1) = 0 + 6 = 6$.

Why does $d(\mathbf{a_1}, \mathbf{a_2})$ measure how adequately the alarms $\mathbf{a_1}$ and $\mathbf{a_2}$ can be modeled by a generalized alarm $\mathbf{g}$? To answer this question, assume that all weights equal one, i.e. $w_i = 1$ for $i = 1, \ldots, n$. Moreover, let $\mathbf{g} \in \mathsf{X}_{1 \le i \le n}\text{Dom}(A_i)$ be any generalized alarm to which both alarms can be generalized, i.e. $\mathbf{a_1}, \mathbf{a_2} \unlhd \mathbf{g}$. Note that $d_i := d(\mathbf{g}, \mathbf{a_i})$, $i = 1, 2$, is the number of times

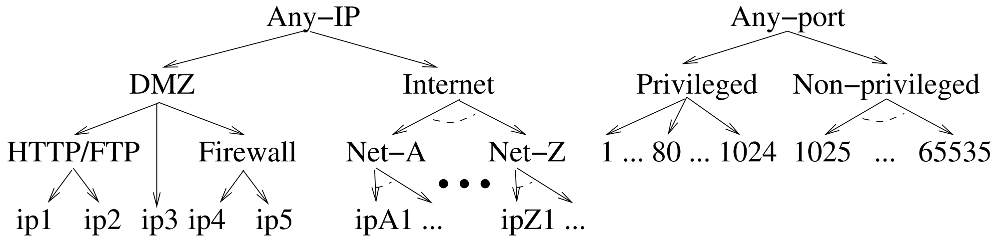
[1]Rada et al. use $\delta(\cdot,\cdot)$ as their distance index [178, 179]. Our preference for $d(\cdot,\cdot)$ over $\delta(\cdot,\cdot)$ is purely methodological. In fact, we consider the index $d(\cdot,\cdot)$ more intuitive, and more consistent with other work that uses generalization hierarchies to define distance [135, 180]. However, the alarm clustering problem as defined in Section 4.2.2 is not affected by whether we choose $d(\cdot,\cdot)$ or $\delta(\cdot,\cdot)$ as our distance index.



a) Network.

Src−IP	Dst−IP	Count
ip1	ip4	1000
ip1	ipA1	1
ip1	ipB1	1
⋮	⋮	⋮
ip1	ipZ1	1
ipA1	ip4	1
ipB1	ip4	1
⋮	⋮	⋮
ipZ1	ip4	1

b) Sample alarm log.



c) Generalization hierarchies for IP addresses and port numbers.

Figure 4.3: Network, alarm log, and generalization hierarchies of the running example.

that an attribute in alarm $\mathbf{a_i}$ must be generalized to transform $\mathbf{a_i}$ into $\mathbf{g}$. Therefore, the smaller the sum $d_1 + d_2$ is, the smaller the number of generalization steps that separate $\mathbf{g}$ from $\mathbf{a_1}$ and $\mathbf{a_2}$, and the more adequate the model $\mathbf{g}$ is for the alarms $\mathbf{a_1}$ and $\mathbf{a_2}$. Conversely, if the sum $d_1 + d_2$ is large, then $\mathbf{g}$ is an inadequate model because it is too abstract, and insufficiently captures the detailed information of the alarms $\mathbf{a_1}$ and $\mathbf{a_2}$. We can therefore use the sum $d_1 + d_2$ to measure the adequacy of $\mathbf{g}$. Given that the distance $d(\mathbf{a_1}, \mathbf{a_2})$ equals the minimum value that the sum $d_1 + d_2$ can possibly assume for any $\mathbf{g}$, we see that $d(\mathbf{a_1}, \mathbf{a_2})$ measures the adequacy of the most adequate generalized alarm. Therefore, a small distance value $d(\mathbf{a_1}, \mathbf{a_2})$ implies that an adequate model exists for the alarms $\mathbf{a_1}$ and $\mathbf{a_2}$.

4.2.2 The Alarm Clustering Problem

In generalization of the distance $d(\cdot, \cdot)$, we now define the *heterogeneity* $H(C)$ of a multi-set C of alarms. Heterogeneity is a function that returns a small value when the multi-set C can be adequately modeled by a generalized alarm. To formally define heterogeneity, let $\mathbf{g}$ be a generalized alarm to which all alarms in C can be generalized, i.e. $\forall \mathbf{a} \in C : \mathbf{a} \trianglelefteq \mathbf{g}$ must hold. The average distance $\bar{d}(\mathbf{g}, C)$ between $\mathbf{g}$ and C, and the heterogeneity $H(C)$ are defined as follows:

$$\bar{d}(\mathbf{g}, C) \quad := \quad (1/|C|) \times \sum_{\mathbf{a} \in C} d(\mathbf{g}, \mathbf{a}) \tag{4.7}$$

$$H(C) \quad := \quad \min\{\, \bar{d}(\mathbf{g}, C) \,|\, \mathbf{g} \in \mathsf{X}_{i=1}^{n} \mathrm{Dom}(A_i),\ \forall \mathbf{a} \in C : \mathbf{a} \trianglelefteq \mathbf{g} \,\} \tag{4.8}$$

Intuitively, average distance measures how adequately the generalized alarm $\mathbf{g}$ models the multi-set C on the average. Moreover, a small heterogeneity value implies that there exists a generalized alarm that models C adequately. A generalized alarm $\mathbf{g}$ with $\forall \mathbf{a} \in C : \mathbf{a} \trianglelefteq \mathbf{g}$ and $\bar{d}(\mathbf{g}, C) = H(C)$ is called a *cover* of C. A cover of C is a maximally adequate model for C. If all generalization hierarchies are trees, rather than DAGs, then there exists exactly one cover for each multi-set C of alarms. Finally, for $C = \{\mathbf{a_1}, \mathbf{a_2}\}$, we have $H(C) = 1/2 \times d(\mathbf{a_1}, \mathbf{a_2})$. The data mining problem to be solved now becomes:

Definition 4.1 *Let L be an alarm log, $min\_size \in \mathbb{N}$ an integer, and $\mathcal{G}_i$, $i = 1, \ldots, n$, a generalization hierarchy for each attribute A_i. The* **alarm clustering problem** *$(L, min\_size, \mathcal{G}_1, \ldots, \mathcal{G}_n)$ is to find a multi-set $C \subseteq L$ that* minimizes *the heterogeneity $H(C)$, subject to the constraint that $|C| \geq min\_size$ holds. We call C an* **alarm cluster** *or* **cluster** *for short.* □

In other words, among all multi-sets $C \subseteq L$ that satisfy $|C| \geq min\_size$, a multi-set with minimum heterogeneity has to be found. If there are several such multi-sets, then any one of them can be picked. Note that the min_size parameter formalizes our hitherto intuitive notion of "largeness", which was first introduced in Proposition 3.1. Finally, once the cluster C has been found, the remaining alarms in $L \smallsetminus C$ can be searched for additional clusters. In particular, one might decide to use a different min_size value for $L \smallsetminus C$, an option that is useful in practice.

In the terms of Chapter 3, alarm clusters approximate alarm groups. While the alarms of an alarm group share (per definition) the same root cause, we have no guarantee that the same holds for the alarms of an alarm cluster. Nonetheless, for a reasonably chosen min_size value, it is justified to believe that most alarms of an alarm cluster do share the same root cause. In fact, the alarm cluster hypothesis (Proposition 3.1) states that root causes generally manifest themselves in alarm clusters. Thus, having found an alarm cluster, it is plausible to postulate that a root cause was at its origin. Chapter 5 continues this discussion and examines ways of verifying whether the alarms of an alarm cluster share the same root cause.

It is interesting to note that from a purely practical point of view, it is probably sufficient when the majority of alarms in an alarm cluster share the same root cause. To see why, just observe that alarm clusters are generally so large that analysts rely on covers (rather than alarm clusters) for root cause analysis. This, however, diminishes the influence that any individual alarm can have on the outcome of the root cause analysis. In particular, if adding or removing an alarm to or from an alarm cluster does not change the cover of the alarm cluster, then this alarm cannot impact a root cause analysis that relies exclusively on the cover.

4.2.3 Definition of Generalization Hierarchies

Our alarm clustering framework assumes that meaningful generalization hierarchies have been defined for all alarm attributes. Figure 4.3(c) shows that such generalization hierarchies exist for IP addresses and port numbers. This section suggests further generalization hierarchies for numerical, time, and free-text attributes. The generalization hierarchies of this section should be seen as examples that demonstrate the usefulness and versatility of the framework. Other generalization hierarchies are possible and might prove useful in practice.

Numerical attributes are dealt with by discretizing them into hierarchically nested intervals. In the simplest case, a human expert manually discretizes each numerical attribute. In doing so, the expert might be bound by regulations, which, for example, might stipulate that "children", "adolescents", "adults", and "senior citizens" are (per definition) people in the age ranges $[1, 10[$, $[10, 20[$, $[20, 65[$, and $[65, \infty]$, respectively. In other cases, the human expert is free to exercise her judgment. For example, suppose that the severity of an alarm is measured by a real number in the range from 1 to 97. Assuming that all severity values in this range are equally likely, the expert might decide that a tree of height four and fan-out four constitutes a useful generalization hierarchy. The leaves of this tree are the values in $[1, 97[$, the next higher level consists of the intervals $[1+6\times i, 7+6\times i[$, $i = 0, \ldots, 15$, followed by the intervals $[1 + 24 \times i, 25 + 24 \times i[$, $i = 0, \ldots, 3$. The interval $[1, 97[$ constitutes the root of the generalization hierarchy.

A drawback of user-defined generalization hierarchies is that they are static. In fact, their inability to adjust to the actual distribution of data values can make them a rather unnatural choice. For example, consider a case where 90% of severity values fall into the range $[1, 20]$. The aforementioned balanced generalization hierarchy is not particularly suitable for this case, because it is too coarse-grained in the range $[1, 20]$, while being too detailed in the range $]20, 97[$. *Dynamic generalization hierarchies*, which are constructed at run-time to fit the actual data distribution, are a pragmatic way to mitigate this shortcoming [52, 89, 139].

In the simplest case, algorithms for the construction of dynamic generalization hierarchies construct hierarchies such that all intervals at a given level of the hierarchy contain the same number of data values [89]. This type of algorithm has the drawback that it assigns close values (e.g. 4 and 5) to distinct intervals, while putting distant values (e.g. 22 and 87) into the same interval, if this is necessary to equalize the number of data values per interval. The resulting generalization hierarchies can be rather counter-intuitive. To mitigate this problem, clustering has been used to find intervals that reflect the natural grouping of the data values [139, 147]. Other algorithms for the construction of dynamic generalization hierarchies are surveyed by Dougherty et al. [52]. This is not the place to explore these algorithms further, nor is it our intention to advocate any one of them. Instead, we conclude that the construction of generalization hierarchies for numerical attributes is a well-understood problem that has many practical solutions.

For time attributes, one typically wishes to capture temporal information such as the distinction between weekends and workdays, between business hours and off hours, or between the beginning of the month and the end of the month. To make the clustering method aware of concepts like these, one can use generalization hierarchies such as the ones in Figure 4.4. For example, the left-hand generalization hierarchy shows that the time-stamp ts_1 can be generalized to the concepts *Sat*, *Weekend*, and ultimately, *Any-day-of-week*. The right-hand generalization hierarchy shows how time-stamps can be generalized according to when during a month they were generated.

It can be desirable to use the two generalization hierarchies of Figure 4.4 simultaneously. To this end, one could combine the two hierarchies into a single one, in which each time-stamp has two parents — one that corresponds to its day of the week and one that corresponds to its day of the month. A drawback of this approach is that it makes the generation of covers more cumbersome. (Recall that covers are the generalized alarms that model alarm clusters, see page 68.) To see why, let C be an alarm cluster whose constituent alarms occur on Monday the second of the month. Unfortunately, the concept "Mon 2nd" is not contained in the combined generalization hierarchy. Therefore, it is not possible to find a cover that has its time attribute set to "Mon 2nd". Several solutions are conceivable:

1. One could accept the loss of information and decide that a cover indicating either "Mon" or "2nd" is good enough.
2. One could generate multiple covers for C, but these covers are mostly redundant and only differ in the values of their time-stamp attributes.
3. One could construct the "full cross-product" of the two generalization hierarchies. The resulting generalization hierarchy contains all concept pairs,

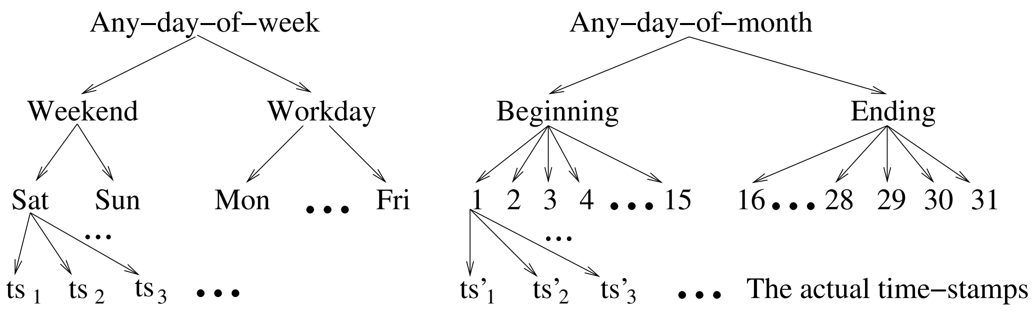


Figure 4.4: Sample generalization hierarchies for time attributes.

including "Mon 2nd". Unfortunately, the generalization hierarchy resulting from a full cross-product is generally very complex.

4. One could replicate the time-stamp attribute and assign each generalization hierarchy to a separate replica. That way, one replica plays the role "day of the week", whereas the other plays the role "day of the month". Further, each replica is generalized according to its own generalization hierarchy.

In practice, alarm clusters are generally huge and cumbersome to work with. Therefore, root cause analysis mostly relies on covers (which model the alarm clusters) rather than on the clusters themselves. Against this background, we consider the first two solutions as undesirable because information-deprived or redundant covers are of reduced practical value. The solutions 3 and 4 are equivalent, but solution 4 is easier to use because it frees the user from the potentially error-prone task of calculating the cross-product of two generalization hierarchies. In fact, solution 4 implicitly calculates the cross-product for the user. In our practical work, we therefore choose solution number 4.

Free-text attributes, such as the context attribute in Section 1.4, assume arbitrary strings as their values. Therefore, the challenge lies in tapping the semantic information of these strings. One way to solve this problem is to use a feature extraction step that precedes the actual alarm clustering. *Features* are crisp bits of semantic information that, once extracted, replace the original strings. Thus, each string is replaced by the set of its features. Note that subset-inclusion defines a natural generalization hierarchy on feature sets. For example, the feature set $\{f_1, f_2, f_3\}$ can be generalized to the sets $\{f_1, f_2\}$, $\{f_1, f_3\}$, or $\{f_2, f_3\}$, which in turn can be generalized to $\{f_1\}$, $\{f_2\}$, or $\{f_3\}$. The next level is the empty set, which corresponds to "Any-feature". Note that the feature extraction process constructs a generalization hierarchy at run-time. Hence, free-text attributes are another instance where dynamic generalization hierarchies are useful.

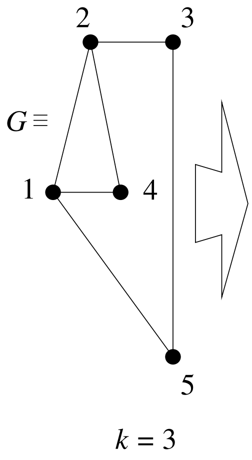
Ultimately, we consider it the intrusion detection analyst's responsibility to select features that capture as much semantic information as possible. In our own work, we found frequent substrings to be useful features. More precisely, let L be an alarm log, let A be a free-text attribute, and let $V := < \mathbf{a}[A] \mid \mathbf{a} \in L >$ be the multi-set of values that attribute A assumes in the alarm log L. We run the Teiresias algorithm [181] on V in order to find all substrings that have a user-defined minimum length and minimum frequency. These substrings are the features and each original string s is replaced by the (single) most frequent feature that is also a substring of s. Thus, all feature sets have size one. Finally, each feature set can only be generalized to the "Any-feature" level. The resulting generalization hierarchy is simple, but frequent substrings have the advantage of being rather understandable features. Therefore, they contribute to the overall understandability of covers.

4.3 Algorithm for Alarm Clustering

This section describes an algorithmic solution to the alarm clustering problem. We begin in Section 4.3.1 by proving that the alarm clustering problem is NP-complete. Given the need for a scalable solution, this constitutes an important result. Moreover, it motivates the remainder of this sections, which describes a heuristic method for alarm clustering. Specifically, Section 4.3.2 describes the classic attribute-oriented induction method, which is the basis of our approach. Section 4.3.3 explains why and how we modified the classic method to make it more suitable for alarm clustering. Section 4.3.4, finally, discusses advanced aspects of the modified attribute-oriented induction method.

4.3.1 NP Completeness of Alarm Clustering

This section proves that the alarm clustering problem is NP-complete. To this end, let L be an alarm log, min_size a minimum cluster size, and $\mathcal{G}_i$ a generalization hierarchy for each attribute A_i in L, $i = 1, \ldots, n$. We show the NP completeness of $(L, min\_size, \mathcal{G}_1, \ldots, \mathcal{G}_n)$ by reducing the CLIQUE problem [166] to the alarm clustering problem. In the CLIQUE problem, which is known to be NP-complete, we are given a graph G and an integer k. The goal is to decide whether G contains a k-*clique*, i.e. a fully connected subgraph of size k. We assume that G contains at least $\binom{k}{2}$ edges because otherwise, there is trivially no k-clique in G. Now, the following steps reduce the CLIQUE problem to the alarm clustering problem (cf. Figure 4.5):



$$L = \begin{array}{c|ccccc} & A_1 & A_2 & A_3 & A_4 & A_5 \\ (1,2) \equiv & (1, & 1, & 0, & 0, & 0) \\ (1,4) \equiv & (1, & 0, & 0, & 1, & 0) \\ (1,5) \equiv & (1, & 0, & 0, & 0, & 1) \\ (2,3) \equiv & (0, & 1, & 1, & 0, & 0) \\ (2,4) \equiv & (0, & 1, & 0, & 1, & 0) \\ (3,5) \equiv & (0, & 0, & 1, & 0, & 1) \end{array}$$

$min\_size = \binom{k}{2} = \binom{3}{2} = 3$

$\forall\ i: G_i \equiv 1 \longrightarrow 0$

Figure 4.5: Example of reducing CLIQUE to alarm clustering.

1. Assign a separate attribute A_i to each node in G. Let $\{A_1, \ldots, A_n\}$ be the resulting set of attributes. Define $\text{Dom}(A_i) := \{0, 1\}$ and $\mathcal{G}_i := 1 \longrightarrow 0$ for $i = 1, \ldots, n$.
2. For each edge e in G, add one alarm $\mathbf{a}$ to L. Let A_i and A_j ($A_i \neq A_j$) denote the attributes that correspond to the endpoints of e. Then, the attributes A_i and A_j of alarm $\mathbf{a}$ are set to 1, whereas all the other attributes are set to 0.
3. In the definition $d(\mathbf{a_1}, \mathbf{a_2}) := \sum_{i=1}^{n} w_i \times d(\mathbf{a_1}[A_i], \mathbf{a_2}[A_i])$ of the distance between two alarms $\mathbf{a_1}$ and $\mathbf{a_2}$, set all weights equal to one, i.e $w_i = 1, \forall i$.
4. Set $min\_size := \binom{k}{2}$, which corresponds to the number of edges in a k-clique.
5. Finally, solve the alarm clustering problem that has just been defined. Let $C \subseteq L$ be an optimal alarm cluster, i.e. an alarm cluster that minimizes $H(C)$ while satisfying $|C| \geq min\_size$. Furthermore, let $\mathbf{c} = (c_1, \ldots, c_n)$ be the cover of C. Then, G contains a k-clique if and only if $\sum_{i=1}^{n} c_i = k$.

To understand step 5, note that the cover $\mathbf{c} = (c_1, \ldots, c_n)$ satisfies $c_i = \max\{\mathbf{a}[A_i] \mid \mathbf{a} \in C\}$ for all i. Moreover, $d(\mathbf{c}, \mathbf{a}) = \sum_{i=1}^{n}(c_i - \mathbf{a}[A_i]) = \sum_{i=1}^{n} c_i - \sum_{i=1}^{n} \mathbf{a}[A_i] = \sum_{i=1}^{n} c_i - 2$ holds for all alarms $\mathbf{a} \in C$, because step 2 guarantees that each alarm has exactly two attributes set to 1. By plugging $d(\mathbf{c}, \mathbf{a}) = \sum_{i=1}^{n} c_i - 2$ into the equations (4.7) and (4.8), we finally obtain $H(C) = \sum_{i=1}^{n} c_i - 2$.

Going back to step 5, suppose that $\sum_{i=1}^{n} c_i = k$ holds. Then, $\{A_i | c_i = 1\}$ defines a k-clique, and the alarms in C correspond to its edges. Conversely, suppose $\sum_{i=1}^{n} c_i > k$, and hence $H(C) > k - 2$ ($\sum_{i=1}^{n} c_i < k$ is impossible). Then, there exists no k-clique in G. Indeed, let us assume that G did contain a k-clique, and let C' be the cluster of alarms that corresponds to the edges of this hypothetical k-clique. Furthermore, let $\mathbf{c}' = (c'_1, \ldots, c'_n)$ be the cover of C'. Now, the equations $\sum_{i=1}^{n} c'_i = k$ and ultimately $H(C') = k - 2$ follow. This contradicts the assumption that $H(C)$ is minimal, while, in fact, $H(C) > H(C')$ holds.

4.3.2 Classic Attribute-Oriented Induction

Given the NP completeness result of the last section, we now describe a heuristic method for solving the alarm clustering problem. For a given problem $(L, min\_size, \mathcal{G}_1, \ldots, \mathcal{G}_n)$, the heuristic method finds an alarm cluster $C \subseteq L$ that satisfies $|C| \geq min\_size$, but does not necessarily minimize the heterogeneity $H(C)$. Obviously, the further a heuristic method pushes $H(C)$ to its minimum, the better it is. The heuristic method developed in this dissertation is a variant of Attribute-Oriented Induction (AOI) [87, 88], a well-known conceptual clustering method, which is described next.

Attribute-oriented induction operates on relational database tables and repeatedly replaces attribute values by more abstract values. These more abstract values are taken from user-defined generalization hierarchies, such as the ones in the Figures 4.3 and 4.4. Thanks to the generalization of attribute values, previously distinct alarms become identical and can be merged. In this way, huge relational tables can be condensed into short and highly comprehensible summary tables.

For a more formal treatment, we extend all alarms by a new integer-valued pseudo-attribute, the so-called *count* (see last column in Figure 4.3(b)). Thus, we henceforth model alarms as tuples over the Cartesian product $\text{Dom}(A_1) \times \ldots \times \text{Dom}(A_n) \times \mathbb{N}$. The count attribute is used by the AOI algorithm for book-keeping, only. The alarm attributes A_i are as before. The inputs of the AOI algorithm are a relational table T over the attributes $\{A_1, \ldots, A_n, count\}$, generalization hierarchies $\mathcal{G}_i$, and generalization thresholds $d_i \in \mathbb{N}$ for all attributes A_i ($i = 1, \ldots, n$). In our case, the table T initially stores the alarms of the alarm log L. For the sake of simplicity, we will assume that all generalization hierarchies are trees.

Figure 4.6 shows the pseudo-code of the classic AOI algorithm. The first step assigns a value of 1 to the count attribute of all alarms. Subsequently, the main loop (steps 2–8) is iterated: Step 3 selects an attribute A_i and the steps 4 and 5 replace the A_i values of all alarms by their parent values in $\mathcal{G}_i$. By doing so, previously distinct alarms can become identical. Two alarms $\mathbf{a}$ and $\mathbf{a}'$ are *identical*

Input: A table T, tree-structured generalization hierarchies $\mathcal{G}_i$, and generalization thresholds d_i;
Output: A generalized table storing generalized alarms;
Algorithm:
1: ***for all*** alarms $\mathbf{a}$ in T ***do*** $\mathbf{a}[count] := 1$; *// Initialize counts.*
2: ***while*** table T is not abstract enough ***do*** $\{$
3: Select an alarm attribute A_i;
4: ***for all*** alarms $\mathbf{a}$ in T ***do*** *// Generalize attribute A_i.*
5: $\mathbf{a}[A_i] :=$ parent of $\mathbf{a}[A_i]$ in $\mathcal{G}_i$;
6: ***while*** identical alarms $\mathbf{a}, \mathbf{a}'$ exist ***do*** *// Merge identical alarms.*
7: Set $\mathbf{a}[count] := \mathbf{a}[count] + \mathbf{a}'[count]$ and delete $\mathbf{a}'$ from T;
8: $\}$

Figure 4.6: Pseudo-code for the classic AOI algorithm.

if $\mathbf{a}[A_i] = \mathbf{a}'[A_i]$ holds for all attributes A_i, while $\mathbf{a}[count]$ and $\mathbf{a}'[count]$ are allowed to differ. The steps 6 and 7 merge identical alarms into a single generalized alarm whose count value equals the sum of individual counts. In this way, the count attribute always reflects the number of original alarms that are summarized by a given generalized alarm. Note that each generalized alarm $\mathbf{a}$ represents an alarm cluster of size $\mathbf{a}[count]$. Moreover, the elements of an alarm cluster $\mathbf{a}$ are the original (ungeneralized) alarms that were merged into $\mathbf{a}$.

One key aspect of the classic AOI algorithm has been left open, namely, how the attributes A_i are selected in step 3. The selection criterion is that any attribute A_i that assumes more than d_i *distinct* values in table T can be selected. (Recall that d_i, the generalization threshold, is an input parameter to the algorithm.) The main loop terminates in step 2 if no such attribute exists. In other words, step 2 considers the table T "abstract enough" when each attribute A_i assumes at most d_i distinct values.

4.3.3 Modified Attribute-Oriented Induction

To summarize, classic AOI generalizes an attribute A_i until it assumes at most d_i distinct values ($i = 1, \ldots, n$). This strategy of bounding the number of distinct attribute values guarantees that the final generalized table contains at most $\Pi_{i=1,\ldots,n} d_i$ generalized alarms. However, this strategy can lead to excessive generalization, in which too much detail is lost (so-called *over-generalization*). This section illustrates the problem of over-generalization and shows how we modified the classic AOI algorithm to mitigate it. For the sake of simplicity, we will assume that all generalization hierarchies are trees, and that all attribute weights in the distance definition $d(\mathbf{a_1}, \mathbf{a_2}) := \sum_{i=1}^{n} w_i \times d(\mathbf{a_1}[A_i], \mathbf{a_2}[A_i])$ equal one, i.e. $w_i = 1, \forall i$. Section 4.3.4 relieves these constraints.

Figure 4.3(b) on page 67 shows a sample table having the alarm attributes Src-IP (the source IP) and Dst-IP (the destination IP). Note that the first tuple in the table represents 1000 occurrences of the alarm $(ip1, ip4)$. We use the left-hand generalization hierarchy of Figure 4.3(c) for both alarm attributes, and we assume that both generalization thresholds have been set to 10 (i.e. $d_1 = d_2 = 10$). Given that both alarm attributes assume 27 distinct values, they are both generalized once by the classic AOI algorithm. This yields a new table whose alarm attributes still have 27 distinct values. Therefore, both attributes are generalized again. The resulting table, which contains the generalized alarms $(DMZ, DMZ, 1000)$, $(DMZ, Internet, 26)$, and $(Internet, DMZ, 26)$, is the final result of classic AOI. Note that this result is (over-)generalized to a point where important details have been lost. In fact, instead of the above result we had rather obtained the alarms $(ip1, ip4, 1000)$, $(ip1, Internet, 26)$, and $(Internet, ip4, 26)$, which are more specific and informative.

A major source of over-generalization is "noise". Indeed, noise forces up the number of distinct attribute values and thereby controls the generalization process. In the above example, there was one main signal (namely the tuple $(ip1, ip4, 1000)$) and five percent of "noise" (the remaining 52 tuples). However, the noise dominated the generalization process and caused the alarm $(ip1, ip4, 1000)$ to be generalized four times, so it became $(DMZ, DMZ, 1000)$. Noise-induced over-generalization is a serious problem in intrusion detection, and it motivates our first modification of the classic AOI algorithm.

Modification 1 *We abandon the generalization thresholds d_i as well as the associated strategy of bounding the number of distinct attribute values. Our new strategy is to find generalized alarms that subsume "many" of the original (un-generalized) alarms. Formally, we search alarms $\mathbf{a} \in T$ that have a count of min_size or larger (i.e. $\mathbf{a}[count] \geq min\_size$), where $min\_size \in \mathbb{N}$ is the user-defined minimum cluster size. Whenever such an alarm is found, it is removed from the table T and reported as a solution to the user. Processing continues with the table $T' := T \smallsetminus \{\mathbf{a}\}$.* □

Recall that each alarm $\mathbf{a}$ represents an alarm cluster of size $\mathbf{a}[count]$. The above modification has two effects: First, by imposing a minimum cluster size of min_size, it forces the AOI algorithm to find "large" clusters. Second, by preventing further generalization of an alarm $\mathbf{a}$ that satisfies $\mathbf{a}[count] \geq min\_size$, it tries to avoid over-generalization. The combined effect is to bias the algorithm towards large clusters that nonetheless have adequate (i.e. specific) models in the form of generalized alarms. Finally, Modification 1 also raises the need for a new attribute selection criteria for step 3 of Figure 4.6. To counteract over-generalization, we use the following heuristic criteria, which tries to minimize the total number of attribute generalizations:

Modification 2 *For each alarm attribute A_i, let $F_i := \max\{f_i(v) \,|\, v \in \mathit{Dom}(A_i)\}$ be the maximum of the function*

$$f_i(v) := \textit{SELECT } sum(count) \textit{ FROM } T \textit{ WHERE } A_i = v,$$

which sums the counts of all alarms $\mathbf{a} \in T$ with $\mathbf{a}[A_i] = v$. Step 3 of Figure 4.6 selects an attribute A_i whose F_i value is minimal, i.e. $\forall j : F_i \leq F_j$ must hold. □

The justification of this heuristic is that an alarm $\mathbf{a}$ with $\mathbf{a}[count] \geq min\_size$ cannot exist unless $F_i \geq min\_size$ holds for all attributes A_i. Therefore, we use it as a heuristic to increase the smallest F_i value by generalizing its corresponding attribute A_i. This heuristic attempts to minimize the number of generalization steps needed to find an alarm $\mathbf{a}$ that satisfies $\mathbf{a}[count] \geq min\_size$. Other heuristics are clearly possible, but our experience supports the one above.

To see what we have achieved so far, let us reconsider the example of Figure 4.3 and let min_size equal 20. The alarm $(ip1, ip4, 1000)$ has a count larger than 20, and is immediately removed from the table and presented to the user. All remaining alarms have counts of one, so that generalization starts. Because of $F_1 = F_2 = 26$, either of the two attributes Src-IP or Dst-IP can be selected for generalization. Without loss of generality, we assume that the attribute Src-IP is chosen and generalized. The resulting alarms still have counts of one, which is why a second generalization step is initiated. Again, we assume that the Src-IP is selected and generalized. The resulting table contains the alarm $(Internet, ip4, 26)$, which is removed and reported to the user. Finally, the Dst-IP attribute is generalized twice, the alarm $(DMZ, Internet, 26)$ is reported, and processing ends.

A problem becomes apparent. Although generalizing the attribute Src-IP has allowed us to find the alarm $(Internet, ip4, 26)$, it has irrevocably over-generalized the source IP of the 26 alarms that remain after $(Internet, ip4, 26)$ was removed. As a consequence, the last alarm reported is $(DMZ, Internet, 26)$, rather than the more specific $(ip1, Internet, 26)$. The problem is that a generalization step can be opportune in the short run while having undesirable late effects. In other words, generalization steps can accumulate to the point where over-generalization occurs. This motivates the next modification.

Modification 3 *After reporting an alarm $\mathbf{a} \in T$ with $\mathbf{a}[count] \geq min\_size$, we used to continue processing with table $T' := T \setminus \{\mathbf{a}\}$. Henceforth, we* first *undo all generalization steps in T'. This involves replacing all generalized alarms by their constituent ungeneralized alarms. Then, processing resumes with the resulting table, in which all counts equal one.* □

Now, let us reconsider the above example: Processing is unchanged up to the point where the alarm $(Internet, ip4, 26)$ is removed from the table. Then,

Input: A table T and tree-structured generalization hierarchies $\mathcal{G}_i$;
Output: Generalized alarms of user-defined minimum sizes;
Algorithm:

1: ***for all*** alarms $\mathbf{a}$ in T ***do*** $\mathbf{a}[count] := 1$; *// Initialize counts.*
2: Prompt the user for a min_size value;
3: ***while*** $\sum_{\mathbf{a} \in T} \mathbf{a}[count] \geq min\_size$ ***do*** $\{$
4:     ***while*** identical alarms $\mathbf{a}, \mathbf{a}'$ exist ***do*** *// Merge identical alarms.*
5:         Set $\mathbf{a}[count] := \mathbf{a}[count] + \mathbf{a}'[count]$ and delete $\mathbf{a}'$ from T;
6:     ***if*** $\exists \mathbf{a} \in T : \mathbf{a}[count] \geq min\_size$ ***then*** $\{$ *// If we found a solution.*
7:         Output the alarm $\mathbf{a}$ and set $T := T \setminus \{\mathbf{a}\}$;
8:         Replace all $\mathbf{a}' \in T$ by their constituent ungeneralized alarms;
9:         Prompt the user for a new min_size value;
10:     $\}$ ***else*** $\{$
11:         Select an attribute A_i as specified by Modification 2;
12:         ***for all*** alarms $\mathbf{a}$ in T ***do*** *// Generalize attribute A_i.*
13:             $\mathbf{a}[A_i] :=$ parent of $\mathbf{a}[A_i]$ in $\mathcal{G}_i$;
14:     $\}$
15: $\}$

Figure 4.7: Pseudo-code for the modified AOI algorithm.

Modification 3 kicks in and resets the Src-IP attribute of the remaining 26 alarms to its original value, which is $ip1$. Finally, the Dst-IP attribute is generalized twice, the alarm $(ip1, Internet, 26)$ is reported, and processing ends. Figure 4.7 summarizes the discussion and shows the modified AOI algorithm. Note that the steps 2 and 9 offer tight control over the min_size values that are used in successive iterations of the main loop.

In Section 3.2.2, we have defined five algorithmic requirements that a data mining technique should satisfy to be suitable for root cause analysis. We can now examine how well the modified AOI algorithm meets these requirements:

Scalability: The excellent scalability of classic AOI [64] is mostly preserved by our modifications. Chapter 6 will present concrete run-time measurements to substantiate this claim.

Noise tolerance: By design, the modified AOI algorithm tolerates the noise typically found in alarm logs (see the discussion before Modification 1).

Ease of use: The min_size parameter has a very intuitive interpretation and is the only one to be set. Some expertise in the application domain is needed to define meaningful generalization hierarchies. However, once defined, these hierarchies are generally static.

Multiple attribute types: As illustrated in Section 4.2.3, AOI can take advantage of a wide variety of attribute types, including numerical, categorical, time, and free-text attributes.

Interpretability & relevance of results: As Chapter 6 points out, we found the results of the modified AOI algorithm to be relevant and easy to interpret.

4.3.4 Advanced Aspects

This section extends the modified AOI algorithm to support DAG-structured generalization hierarchies. In addition, it shows how attribute weights w_i different from one can be handled, i.e. the case $d(\mathbf{a_1}, \mathbf{a_2}) := \sum_{i=1}^{n} w_i \times d(\mathbf{a_1}[A_i], \mathbf{a_2}[A_i])$ with $w_i \neq 1$ is considered. Finally, we discuss the problem of setting the parameter min_size.

To begin with, when generalization hierarchies are DAGs rather than trees, then any node can potentially have multiple parents. Consequently, there is no longer a unique parent that an attribute value can be generalized to. There are two basic strategies for resolving this issue:

Choose-one: This strategy employs user-defined rules to resolve ambiguities. For example, consider an IP address $ip1$ that simultaneously runs an HTTP server and an FTP server. Accordingly, *HTTP-server* or *FTP-server* are two possible generalizations of $ip1$. The following rule assumes that $ip1$ is the destination IP of alarm $\mathbf{a}$, and generalizes $ip1$ according to the value of the destination port:

> ***if*** $(\mathbf{a}[Dst\text{–}port] = 80)$ ***then*** generalize $ip1$ to *HTTP-server*
> ***else*** generalize $ip1$ to *FTP-server*;

A similar rule is conceivable for the case that $ip1$ is the source IP address of an alarm.

Explore-all: This strategy pursues all possible generalizations in parallel and retains the one that first leads to a generalized alarm of size min_size or larger.

Both of these strategies have been studied in the context of classic AOI [35, 86], and we can directly reuse the solutions proposed there. In the choose-one strategy, the problem of *induction anomaly* needs to be solved [35]. Induction anomalies arise when a user-defined rule tries to access attribute values that have been "abstracted away" in a previous generalization step. For example, the above rule for the HTTP/FTP server is only applicable when the destination port number has not previously been generalized. A solution to the induction anomaly problem is described in [35], and we have adopted this solution in our implementation of the modified AOI method. We have not implemented the explore-all strategy, because the proposed exhaustive search seems incompatible with our scalability requirement. However, the explore-all strategy is conceptually easy to implement [86]: First, line 13 of Figure 4.7 is replaced by:

13.1: T := T $\setminus \{\mathbf{a}\}$;
13.2: ***for all*** parents p that $\mathbf{a}[A_i]$ has in $\mathcal{G}_i$ ***do*** {
13.3: $\mathbf{a}' := \mathbf{a}$; $\mathbf{a}'[A_i] := p$; $T := T \cup \{\mathbf{a}'\}$;
13.4: }

In other words, the attribute A_i of alarm $\mathbf{a}$ is generalized in all possible ways and the resulting alarms $\mathbf{a}'$ are added to T. Now, however, the clusters modeled by the generalized alarms in T are no longer disjunct. It is therefore incorrect to merge generalized alarms by adding their counts as is done in line 5 of Figure 4.7. The easiest way to determine the correct count values of generalized alarms is to rescan the original alarm log and to determine the number of original alarms that match them. More efficient implementations are possible. Finally, because the clusters in T might overlap, it is also necessary to implement step 8 more carefully.

Another generalization over the previous section is to add support for attribute weights $w_i \neq 1$. As explained in Section 4.2, a large attribute weight w_i penalizes the generalization of the corresponding attribute A_i, $i = 1, \ldots, n$. The best place to incorporate this bias into the approximation algorithm is in step no. 11 of Figure 4.7, where an attribute is selected for generalization. Different heuristics are possible to make this step aware of weights. We have implemented a simple extension of the heuristic described in Modification 2. Specifically, if all attributes A_i satisfy $F_i \geq min\_size$, then we multiply the F_i by the weights w_i, i.e. we set $F_i := F_i \times w_i$, $i = 1, \ldots, n$. Otherwise, we leave the F_i unchanged. Next, among all attributes that have not yet been generalized to the root of their hierarchies, we select an A_i with a minimal F_i value, i.e. $\forall j : F_i \leq F_j$ must hold. As intended, this heuristic is biased towards selecting attributes with small associated weights.

Our last extension over the previous section concerns the min_size parameter. Currently, the user is the only authority over this parameter. If the user chooses an excessively large min_size value, then the quest for a cluster of at least this size can force the clustering algorithm to merge alarms with different root causes. This is undesirable because the resulting alarm clusters can be hard or even misleading to interpret. On the other hand side, if min_size is set to an overly small value, then clustering can end prematurely and alarms that have the same root cause can end up in different clusters. This can inflate the number of clusters and the time needed to interpret them. To mitigate these problems, it seems desirable to give the user some assistance in setting the min_size parameter.

Before investigating ways of setting the parameter min_size, it is important to note that other clustering methods have very similar parameters (see Section

4.1.2). For example, partitioning methods require the user to specify the *number of clusters* that the dataset is supposed to contain. Similarly, density-based methods expect the user to define the *minimum number of data objects* that a data volume must contain to count as "dense". In general, most clustering methods have parameters like these, which allow the user to control the desirable amount of clustering. Intuitively, these parameters select an "operating point" between the two extremes of no clustering (i.e. all objects form clusters by themselves) and maximum clustering (i.e. all objects are grouped into a single cluster). Aside from a few ad hoc rules [9, 104], there exist no guidelines for choosing the operating point.

We now describe the revised scheme for setting the min_size parameter. As before, the user is prompted in the lines 2 and 9 of Figure 4.7 to provide a min_size value for the next cluster. However, this value is no longer used as is, but rather serves as the seed value of an iterative process that converges towards a robust min_size value. This robust value is finally used for clustering. Intuitively, a min_size value is *robust* if slightly smaller or slightly larger values still yield the same clustering result. Robustness is an important property because it limits the effects that spurious or arbitrary decisions can have on the clustering results.

Formally, let ε be a small fraction of 1, e.g. $\varepsilon = 0.07$. A given min_size value ms is *ε-robust* if it yields the same alarm cluster as the values $(1 - \varepsilon) \times ms$ and $(1 + \varepsilon) \times ms$. We can test for ε-robustness by simulating the alarm clustering method for the values ms, $(1 - \varepsilon) \times ms$, and $(1 + \varepsilon) \times ms$. The value ms is ε-robust if all three simulations yield the same alarm cluster. Note that ε-robustness is always relative to a particular alarm log. Thus, a given min_size value can be ε-robust with respect to one alarm log, while being in-robust for another alarm log.

Let ms_0 be the min_size value that the user originally keyed in. This value is used for clustering if and only if it is ε-robust (with respect to the alarm log at hand). Otherwise, it is depreciated according to the formula $ms_{i+1} := (1 - \varepsilon) \times ms_i$. This test-and-depreciate cycle is iterated until an ε-robust min_size value has been found. In the worst case, termination occurs after $O(\log(ms_0))$ iterations. Our decision to progressively decrease rather than increase the min_size value stems from our desire not to mix alarms of different root causes in the same alarm cluster. Section 3.2 and Chapter 5 discuss this point in great detail.

4.4 Discourse on Background Knowledge

The classic and the modified AOI methods use background knowledge in the form of generalization hierarchies to decide which clusters to form. This section explores the integration of other kinds of background knowledge into the clustering process. The expected benefit of adding more background knowledge is to obtain potentially better clustering results. In particular, one might hope to get closer to the ideal (but unattainable) situation where the alarms of an alarm cluster are guaranteed to share the same root cause.

The remainder of this section describes three different types of background knowledge, and discusses how each applies to alarm clustering. Without exception, we found that a substantial research or development effort would have been necessary to reap the benefits from these types of background knowledge. We have therefore not implemented and experimented with any of them. Doing so could be the subject of future research.

4.4.1 Contextual Background Knowledge

Remember that the alarms of an alarm cluster should ideally share the same root cause. A necessary prerequisite for obtaining such clusters is that alarms contain enough information, so that in theory, one can decide if alarms have the same root cause. As will be shown next, the alarms of network-based IDSs do not meet this prerequisite.

For example, consider two alarms that have the same source IP address. Because of DHCP (Dynamic Host Configuration Protocol) and NAT (Network Address Translation) firewalls, these two alarms might be triggered by completely different machines that have nothing in common. Similarly, the dial-in routers of ISPs (Internet Service Providers) assign IP addresses dynamically, so that an IP address allows no conclusion to be drawn about the machine attached to it. Finally, even when two alarms are known to originate from the same source machine, they might still have been caused by completely independent users or processes. To summarize, alarms do not unambiguously identify the actors (i.e. machines, users, and processes) involved in their generation. As a consequence, it is not possible to reliably decide if different alarms have the same root cause.

An obvious remedy is to collect *contextual background knowledge*, such as DHCP logs, NAT logs, and other log files. Using this background knowledge it is possible to unify alarms before they are processed by the alarm clustering method. For example, unification can replace all IP addresses by unique tokens, which unambiguously identify the respective machines. This would solve some of the problems associated with naming ambiguity. Unfortunately, collecting, maintaining, and processing the necessary contextual background knowledge constitutes a major development endeavor, which exceeds the scope of this dissertation.

4.4.2 Feature Construction

Feature construction is the process of extending a given dataset by adding new attributes that are deemed to be relevant for improving the data mining results. The new attributes can be set according to prior knowledge, an analysis of the already existing attributes, or a combination of both. For example, the MADAM ID of Section 2.3.3 extends connection records by new attributes that store statistical summaries over chronologically close connection records. Analogously, one can attempt to construct new attributes that differentiate alarms with supposedly different root causes. Possible candidates are:

- A new attribute that indicates if the current alarm is part of a periodic pattern, in which a given source IP address triggers an alarm every so-and-so many seconds.
- A new attribute that indicates if the current alarm is part of a host scan, in which a given source IP address probes many different destination IP addresses.
- A new attribute that indicates if the destination IP address of the current alarm is an uncommon and supposedly suspicious target.

To understand the rationale behind the first of the above three candidate attributes, let $\mathbf{a}_{period}$ be an alarm that belongs to a periodic pattern, and let $\mathbf{a}_{none}$ be an alarm that does not. In general, one would assume that the alarms $\mathbf{a}_{period}$ and $\mathbf{a}_{none}$ have different root causes and should therefore not be clustered. Adding a new attribute, which is set to "1" in $\mathbf{a}_{period}$ and to "0" in $\mathbf{a}_{none}$ enforces this intuition. In fact, the two alarms might still be merged into a single cluster, but only at the expense of an additional generalization step. Depending on the weight that is assigned to this generalization step, its cost can be so high that the algorithm chooses not to perform it, thus leaving the two alarms in separate clusters. Similar arguments can be made to justify the other two candidate attributes.

Feature construction is no panacea. Specifically, in addition to the above three candidate attributes, there are many other attributes that are equally plausible. Simply using all of them is problematic, because the quality of clustering results has been shown to decrease when irrelevant attributes (called *noise-dimensions*) are added [148, 154]. Therefore, the problem of *feature selection*, i.e. of selecting the "best" attributes, arises. However, most research has focused on selecting features for classification problems, rather than clustering problems [58, 193]. Therefore, selecting features that are beneficial to alarm clustering is difficult, and research into feature selection is beyond the scope of this dissertation.

4.4.3 Constraint-Based Clustering

Assume we knew that certain alarms have (or have not) the same root cause. It would be desirable to make the clustering method aware of these constraints so that it can enforce them, and only return clusters that are consistent with them. *Constraint-based clustering methods* have the ability to do exactly that [65, 81, 194, 200]. Usually, the constraints supported by constraint-based clustering methods are relatively simple:

- Wagstaff and Cardie support *must-link* and *cannot-link* constraints [200]. A must-link constraint specifies that two data objects must end up in the same cluster, while a cannot-link constraint enforces their separation.
- Talavera and Béjar allow the user to specify predicates, and enforce that all data objects satisfying the same predicate end up in the same cluster [194].
- A *contiguity constraint* is a connected graph $\mathcal{G}$ over the dataset under investigation. A contiguity constraint $\mathcal{G}$ enforces that only clusters are returned whose constituent data objects induce a connected subgraph over $\mathcal{G}$. For example, when clustering geographic regions based on similarities in their animal populations, one can use contiguity constraints to enforce that clusters consist of geographically adjacent regions [65, 81].

We anticipate three problems when applying the constraint-based idea to alarm clustering: First, constraints that are useful for alarm clustering need to be identified. For example, defining must-link and cannot-link constraints for pairs of individual alarms does not seem practical given the size of alarm logs (cf. Table 1.2). Second, once a class of constraints has been identified, we are doubtful if the background knowledge needed to instantiate it was available. Third, if the background knowledge was available, then we hypothesize that it would be better used by an expert system that groups alarms as it is told. This, however, carries us away from the focus of this dissertation, which is the discovery of new and unexpected root causes. Expert systems, by contrast, search for old and well-known root causes.

Chapter 5

Cluster Validation

Ideally, we would like to prove our alarm clustering method to be correct and complete in the sense of Definition 3.2. However, over the past 50 years, there has been no precedence of a similar proof. *Cluster validity* is the commonly accepted weaker alternative to proofs of correctness or completeness. Section 5.1 defines cluster validation and further explains why it is needed. Section 5.2 surveys the field of cluster validation. Section 5.3 explains the practical limitations of cluster validation, and describes the validation approach that we have pursued in our alarm clustering framework.

5.1 The Validation Dilemma

Clustering has traditionally been used to explore new datasets whose properties were poorly understood. Therefore, the principal goal of clustering was to uncover the hidden structures in a dataset, and to stimulate theories that explain them. Accordingly, there were claims that the main criterion for assessing a clustering result was its interpretability and usefulness. However, clustering methods have important limitations that make such an ad hoc approach to cluster validation a dangerous endeavor:

The random data phenomenon: Clustering methods *always* find clusters, no matter whether they exist or not. In fact, even when applied to random data, they mechanically group it into clusters. Clearly, such clusters are meaningless at best, and misleading at worst.

Imposed structures: Clustering methods are not "neutral", but have a tendency to *impose* a structure on the dataset [5, 9, 81, 83]. For example, the K-means method is known to favor spherical (a.k.a. globular) clusters of approximately equal sizes [81, 83, 90]. When a dataset contains clusters of a different kind (e.g. clusters that form parallel lines in the plane), then the K-means method is nonetheless inclined to impose "synthetic" clusters of globular shapes and similar sizes. Clearly, these clusters are an artifact of the K-means method, and do not reflect the true structure of the dataset.

Method-dependent results: A corollary to the last point is that markedly different results can be obtained when a given dataset is analyzed by different clustering methods [5, 9, 56, 81].

Given these drawbacks, and given that the human mind is quite capable of providing post hoc justification for clusters of dubious quality, there are clear dangers in a purely manual approach to cluster validation. Specifically, the risk of inadvertently justifying and approving meaningless clusters can be high [56, 81, 104]. Note that a priori, these reservations also apply to the clusters found by our alarm clustering method. We would therefore like to prove the correctness and completeness (as defined in Definition 3.2) of our clustering method, but such proofs do not exist [9, 104, 148, 153]. Cluster validation offers a weaker alternative according to which a cluster structure is *valid* if it cannot reasonably be explained by chance or by the idiosyncrasies of a clustering method. In practice, a cluster structure is declared valid, if there is evidence that it correctly captures the natural groups of the underlying dataset. The *cluster structures* under consideration are hierarchies, partitions, and individual clusters (cf. Section 4.1.2).

As will become evident in the next section, cluster validation is a completely formal endeavor, that is rid of any notions of semantics. This can lead to the situation where a cluster structure is declared "valid" even though it has no meaningful interpretation. For example, Figure 5.1 plots for lawyers, engineers, and students their respective wine consumptions at various wine prices. Clearly, the two ellipsoidal clusters capture the structure of the dataset rather well. Therefore, these clusters are valid by most validity measures, even though they have no obvious interpretations (at least not in terms of peoples' professions). This illustrates that cluster validation can at best increase our confidence in a cluster structure, but it cannot proof the structure to be correct or meaningful [5, 57, 80, 85].

5.2 Cluster Validation Background

Cluster validation can be used to validate hierarchies, partitions, or individual clusters [54, 80, 81, 104]. To keep the presentation simple, this section only considers the validation of partitions (i.e. sets of non-overlapping clusters). The decision to focus on partitions rather than hierarchies or individual clusters was taken because our alarm clustering method produces partitions. Moreover, hierarchies and individual clusters are validated in basically the same manner.

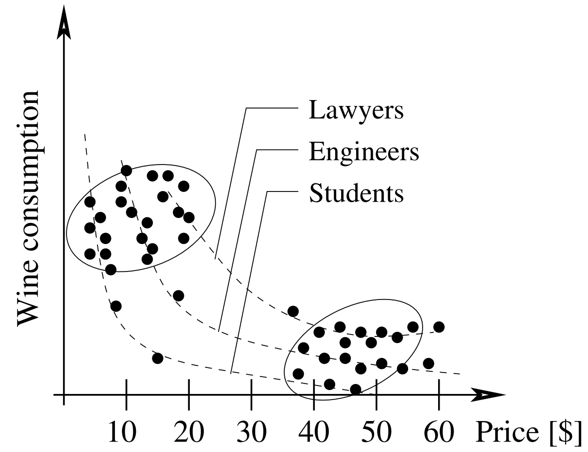


Figure 5.1: Example of valid clusters that have no intuitive interpretation.

Very briefly, the validation techniques of this section can be characterized as follows: The techniques of Section 5.2.1 declare a partition valid if it is “unusually good” under the circumstances of the study. Section 5.2.2 argues that a partition is valid if essentially the same partition is obtained by clustering an independently drawn dataset that follows the same distribution. The Sections 5.2.3 and 5.2.4 evaluate clustering methods rather than their results. Here, the goal is to demonstrate certain favorable properties that make a clustering method more likely to produce valid clusters. Section 5.2.5 describes more ad hoc validation techniques that were born out of practice. Note that this section is about cluster validation in general. Validation of alarm clusters is addressed in Section 5.3.

5.2.1 Cluster Validation Based on Statistics

In a statistical sense, a partition is valid if it is “so good” that it is unlikely to be the result of chance [54, 56, 80, 104]. Two different criteria have been formulated to measure the “goodness” of a partition:

External criteria measure goodness with respect to some externally provided ground truth. Typically, the user defines ground truth to be the partition that she considers to be correct or desirable. Then, goodness is the degree of match between this ideal partition and the actual one.

Internal criteria consider a partition to be good, if it fits the dataset it was derived from. For example, a partition that organizes a dataset into compact and isolated clusters fits the dataset well, and is commonly judged to be good.

An *index* is a formula that measures goodness in a quantitative manner. External criteria use indices that measure the degree of match between two partitions (namely the ideal partition and the partition under evaluation). Internal criteria, on the other hand side, use indices that measure the fit between a partition and the dataset that the partition was derived from.

It is relatively easy to define indices, but it is very hard to interpret their return values. Specifically, what does a return value of, say, 0.7 mean? Is that evidence enough to declare a partition valid? Statistics offers a principled framework to answer these questions. The basic idea is to take an index and to study its return values when applied to random partitions. Then, a new partition is valid if the index returns a value that was never (or seldom) observed for random partitions. That is, a partition is valid, if the assigned index value is unlikely for random partitions. For a more detailed discussion, we next consider external and internal criteria separately.

External criteria: Let C and O be two partitions of the dataset D, and let n be the size of D (i.e. $n = |D|$). We assume that the partition C is the result of a clustering method, while the partition O comes from an oracle that knows the "right" way to partition the dataset D. External criteria declare the partition C valid if C and O match unusually well. The practical question of where the partition O comes from is not considered by external criteria.

Formally, the validity of C with respect to O is tested in several steps. The first step is to choose an index that measures the degree of match between the two partitions. Many indices have been suggested for this purpose [104]. For example, the Rand index measures the fraction of object pairs that are classified consistently by both partitions [101]:

$$R(C, O) := |\{\, (\mathbf{x}, \mathbf{y}) \in D \times D \mid \mathbf{x} \neq \mathbf{y} \wedge I_C(\mathbf{x}, \mathbf{y}) = I_O(\mathbf{x}, \mathbf{y}) \,\}| \,/\, \binom{n}{2} \tag{5.1}$$

where $I_C(\mathbf{x}, \mathbf{y})$ is the indicator function for partition C (i.e. $I_C(\mathbf{x}, \mathbf{y}) := 1$ if $\mathbf{x}$ and $\mathbf{y}$ belong to the same cluster of C, and $I_C(\mathbf{x}, \mathbf{y}) := 0$, otherwise), and $I_O(\mathbf{x}, \mathbf{y})$ is defined analogously. Details are given elsewhere [101, 104].

Once an index has been selected, it is used to compare the partitions C and O. The index returns a numeric measure of match. We will assume without loss of generality that large index values indicate good matches. As previously explained, external criteria declare the partition C valid if it matches O "unusually well", i.e. if the index value for C and O is "unusually large". To be "unusually large", the index value must at least be significantly larger than the index values achieved by random partitions of the same dataset.

The notion of "random partition" is formally defined by means of a *randomness* or *null hypothesis*. Different randomness hypothesis are possible [104]. For example, the *random label hypothesis* [104] states that random partitions R are obtained by randomly relabeling the objects of O. For example, if O equals $\{\,\{1,2,3\},\ \{4,5\},\ \{6\}\,\}$ and $\pi(\cdot)$ is a permutation of the numbers 1 to 6, then $R := \{\,\{\pi(1),\pi(2),\pi(3)\},\ \{\pi(4),\pi(5)\},\ \{\pi(6)\}\,\}$ is a valid random partition under the random label hypothesis. Note that there are $n!$ possible random partitions.

Having selected an index $\mathcal{I}(\cdot,\cdot)$ and a randomness hypothesis H_0, we determine the distribution of $\mathcal{I}(C,\cdot)$ under H_0 (known as *baseline distribution*). This distribution shows for all x in the range of $\mathcal{I}(\cdot,\cdot)$ the probability that $\mathcal{I}(C,R)$ equals x, when R is a random partition as defined by H_0. In general, it is not possible to determine the distribution of $\mathcal{I}(C,\cdot)$ under H_0 analytically. Therefore, one might have to approximate it by evaluating $\mathcal{I}(C,R)$ for a large number of random partitions R. Given the distribution of $\mathcal{I}(C,\cdot)$ under H_0, we are ready to decide whether the value $\mathcal{I}(C,O)$ is "unusually large" (or, equivalently, whether the partition C is valid): Let α be a small number, such as 0.01. The value $\mathcal{I}(C,O)$ is "unusually large" if $\mathcal{I}(C,\cdot)$ under H_0 has a probability of α or less to be larger than $\mathcal{I}(C,O)$. In this case, the partition C is called valid at significance level α.

Internal criteria: Internal criteria measure if a partition is intrinsically appropriate for the dataset it was derived from. No external comparative measure such as the above partition O is used. Formally, internal criteria use indices that measure the degree of fit between a partition and the dataset itself. If this fit is "unusually good", then the partition is declared valid. As above, the notion of "unusually good" is defined in a statistical sense: The fit is "unusually good", if it is significantly better than the fit between random data and its best partitioning.

Many indices have been suggested to measure the goodness of fit between a partition and its underlying dataset [149]. For example, let D be a dataset, $d(\cdot,\cdot)$ a distance index on the objects of D, and C a partition of D. The Goodman-Kruskal γ statistic [77, 149] is defined as $\gamma := (S_+ - S_-)/(S_+ + S_-)$, where S_+ (S_-) counts the number of quadruples $(\mathbf{x},\mathbf{y},\mathbf{u},\mathbf{v}) \in D^4$ for which $\mathbf{x}$ and $\mathbf{y}$ belong to the same cluster of C, $\mathbf{u}$ and $\mathbf{v}$ belong to different clusters, and $d(\mathbf{x},\mathbf{y}) < d(\mathbf{u},\mathbf{v})$ ($d(\mathbf{x},\mathbf{y}) > d(\mathbf{u},\mathbf{v})$, respectively). Note that $\gamma \in [-1,1]$, with $\gamma = 1$ for perfect fit.

For the further discussion we assume an index $\mathcal{I}(\cdot,\cdot)$ that returns large values for good fit. Then, a partition C of the dataset D is valid if the value $\mathcal{I}(C,D)$ is "unusually large", i.e. if it is significantly larger than the index values typically obtained for partitions of random data. As before, "random data" is defined by specifying a randomness hypothesis. For example, random data can be the result of garbling the dataset D. Alternatively, objects that were randomly placed in a region of p-dimensional space can serve as random data. Many other randomness hypotheses are possible, and choosing an adequate one is a difficult and poorly understood problem [79, 80].

The value $\mathcal{I}(C, D)$ is "unusually large" (and the partition C is valid) if the following holds for a small user-defined number $\alpha \in [0, 1]$: Let D_R be a random dataset that conforms to the adopted randomness hypothesis, and let C_R be the result of partitioning this dataset with the same clustering method that produced C. The partition C is valid, if the probability of $\mathcal{I}(C_R, D_R)$ being larger than $\mathcal{I}(C, D)$ is smaller than α. As before, few theoretical results exist, and simulation studies are necessary to estimate the probability $P(\mathcal{I}(C_R, D_R) > \mathcal{I}(C, D))$.

5.2.2 Replication Analysis

Replication analysis [27, 145, 156] is based on the logic of cross-validation [96] as used in regression studies. Specifically, a cluster structure is declared valid if it is consistently discovered across different samples that are independently drawn from the same general population. Replication analyses comprise five major steps:

1. Two samples are required to conduct a replication analysis. Perhaps the most direct way to accomplish this is to randomly divide one larger dataset into two samples.
2. A clustering method is chosen, and the first sample is partitioned using this method.
3. The second sample is partitioned using the same clustering method as in step 2.
4. The second sample is re-partitioned by assigning each of its data objects to the most similar (according to some measure) cluster in the partition of the first sample. This yields another partition of the second sample.
5. The Rand index (cf. equation (5.1)) or some other measure of partition agreement is used to compare the two partitions of the second sample. The greater the level of agreement between the two partitions, the more confidence one may have in their validity.

Replication analysis assesses the stability of a clustering result with respect to sampling-related changes in the dataset. Stability is an important property, because it gives us some confidence that a cluster structure is robust and not an artifact of chance or an idiosyncrasies in the experimental setup. Other approaches for testing stability exist. For example, Gnanadesikan et al. assess the stability of a clustering structure when noise is added and when data objects are removed [76].

5.2.3 Simulation Studies

While the last two sections have shown how to validate the results of a clustering method, we now try to validate clustering methods themselves. The basic idea is to determine the intrinsic properties of clustering methods. Then, given a specific clustering problem, the hope is that one can choose an intrinsically suitable clustering method, i.e. a method that yields valid results by construction. Obviously, this approach hinges on our ability to determine the intrinsic properties of a clustering method. Simulation studies are one way of ascertaining these properties.

Simulation studies (a.k.a. Monte Carlo studies) [148, 150, 151, 154] assess clustering methods by their ability to correctly cluster artificially generated datasets. More precisely, a simulation study comprises three major steps. First, artificial datasets with known cluster structures are generated. Second, these datasets are analyzed by the clustering methods of interest to the study. Third, the level of agreement between the known cluster structures and the structures recovered by the various methods is determined by means of one or more recovery indices, such as the Rand index of equation (5.1). Clustering methods that recover the "correct" cluster structures (i.e. the cluster structures that were intentionally embedded in the artificial datasets) are deemed valid for the kind of datasets used in the study.

As simulation studies generate their own datasets, there is no doubt as to the true cluster structure, or the extent to which any given clustering method has recovered this structure. Because of the use of computers to generate the datasets, the results of simulation studies can be based on hundreds or even thousands of analyses. One of the most important results of simulation studies is that the performance of most clustering methods is strongly dependent on the characteristics of the datasets they are applied to [5, 148, 150, 154]. In fact, the number of clusters in the dataset, their absolute and relative sizes, the degree of cluster overlap, as well as various noise conditions are only a few factors that are known to influence the comparative performance of clustering methods. This clearly shows that there is no single "uniformly best" clustering method.

The main disadvantage of simulation studies is their limited generalizability [151, 154]. That is, the clustering method that ranked best in a simulation study is only optimal for the very specific kind of datasets used in the study. In particular, said clustering method might not be optimal for a dataset that contains a different type of noise or a different number of clusters. As a consequence, if one intends to choose a clustering method based on its ranking in a simulation study, then one needs to know the precise characteristics of the dataset to be analyzed. In practice, these characteristics are hardly ever known, which limits the practical value of simulation studies.

5.2.4 Analytic Cluster Validation

Aside from using simulation studies, researchers have also tried to analytically explore the characteristics of various clustering methods. As in Section 5.2.3, the motivation for this work was to better understand clustering methods, and to offer well-founded guidelines for the choice of a method that performs well (i.e. is "valid") in a given applied cluster analysis.

Fisher and Van Ness [69, 161] evaluate clustering methods with respect to a list of intuitively appealing properties, called *admissibility criteria*. The idea is that admissibility criteria should be such that any "reasonable" clustering method satisfies them. Examples of admissibility criteria are given below:

- A clustering method is *convex admissible* if it is guaranteed to find clusters that have non-intersecting convex hulls.
- A K*-group clustering* consists of K clusters such that all within-cluster distances are smaller than all between-cluster distances. A method is K*-group admissible* if it finds K-group clusterings whenever they exist.
- A clustering method $\mathcal{M}$ is *cluster omission admissible* if the following is guaranteed for all datasets D: The partition $\mathcal{M}(D)$ obtained by clustering the dataset D with method $\mathcal{M}$ satisfies $\mathcal{M}(D) = \mathcal{M}(D \setminus C) \cup \{C\}$ for any cluster $C \in \mathcal{M}(D)$, i.e. the objects in $D \setminus C$ are clustered in exactly the same way, whether the objects in C are present or not.
- A method is *point proportion admissible* if the x-fold duplication of any object of the input dataset does not affect the clustering result ($x \in \mathbb{N}$).

Fisher and Van Ness present tables that evaluate nine clustering methods with respect to eleven admissibility criteria. To see how such tables are used, consider a dataset that resulted from a flawed sampling process, in which objects are inadvertently replicated. In this case, a point proportion admissible method is advisable because it is unaffected by potential duplicates. Note that admissibility criteria can only eliminate unsuitable methods, but they cannot indicate a single best method.

Other theoretical work comes from Jardine and Sibson [108] who propose an elegant axiomatic framework for defining an acceptable clustering method. For example, they require that permutation of the objects or multiplication with a constant scale factor should not change the clustering in an essential way. Taken together, their axioms are only satisfied by the single-link method (see p. 57). The major criticism of this work has centered on the axiom that requires classifications to be (in a mathematical sense) continuous functions of the dataset [39]. References on further work in analytic cluster validation are given in [57, 153].

5.2.5 Other Validation Approaches

This section briefly surveys several somewhat ad hoc approaches to cluster validation. Most of these approaches were born out of practice and have not been systematically studied in the academic literature.

Dubes and Jain point out that rather than searching for a rigorously objective measure of cluster validity, it is more practical to validate a cluster structure by applying several different clustering methods and checking if the results agree [56, 57]. Anderberg [9] and Gordon [81] also recommend to analyze a given dataset using more that one clustering method. Moreover, Gordon shows how the results obtained from different methods can be synthesized into a single consensus result. *Relative criteria* [81, 104], by contrast, keep the clustering method fixed, but vary the values of its parameters. Then, heuristic arguments [152] are used to determine the parameter values that produced the comparatively best result.

A variant of external criteria as presented in Section 5.2.1 leaves some attributes out during clustering and uses them subsequently for validation [5, 66, 151]. According to this approach, clusters are valid if they differ significantly on the attributes that were not used during the cluster analysis. One reason this approach is not used frequently is that most researchers find it difficult to omit potentially relevant attributes from the cluster analysis.

Probably the most common validation strategy has been to verify clustering methods on artificial datasets, usually two-dimensional, for which the goodness of the method can be judged visually. Typical examples of such analyses are found in [61, 83, 115, 203].

5.3 Validation of Alarm Clusters

Section 5.1 explained that human beings can be careless and subjective when interpreting alarm clusters, or any other kind of clusters for that matter. Consequently, the importance of objective measures of cluster quality was emphasized. Such measures are studied in the field of cluster validation, which was the topic of Section 5.2. The present section investigates how cluster validation can be applied to the alarm clustering method of Chapter 4. In a first step, Section 5.3.1 highlights the limitations of cluster validation techniques. Section 5.3.2 presents the validation techniques that we adopted, and Section 5.3.3 contains some concluding remarks.

5.3.1 Limitations of Cluster Validation

After Section 5.2, it should be no surprise that cluster validation is no panacea. In fact, cluster validation has hardly been used in practice [5, 80] and over the past 20 years, researchers have continuously emphasized the need for more research in cluster validation [5, 56, 57, 80, 104]. The specific limitations of the various approaches to cluster validation can be summarized as follows:

External criteria: *(i)* In general, computer simulations are needed to approximate the distribution of the index under the randomness hypothesis. Such simulations have *high computational costs* [104]. *(ii)* External criteria establish validity by measuring the fit between the clustering found by a clustering method and the "correct" clustering as provided by an oracle. In practice, there are no oracles, and *the correct clustering is hard to obtain. (iii)* External criteria only check if one specific dataset was clustered correctly. This offers no insight into how well other datasets are clustered. Thus, the results obtained by an external criterion are of *little generality*.

Internal criteria: *(i)* Internal criteria necessitate computer simulations, as well. Therefore, they also have *high computational costs* [104]. *(ii)* The index and randomness hypothesis chosen are known to significantly influence validation results [23, 79, 149]. However, few guidelines exist, so that these *key parameters are generally selected in an ad hoc manner.* This is deeply unsatisfying given the importance of these parameters. *(iii)* Internal criteria offer *weak evidence for the validity for a cluster structure*. For example, a cluster structure that imposes four clusters on a dataset having six true clusters, is likely to be rated as valid by internal criteria [104].

Replication analysis: Replication analysis is certainly one of the *most poorly researched validation methods.* In fact, the meaningfulness of a replication analysis depends on characteristics of the dataset at hand, as well as on design decisions made during the replication analysis [5, 27, 145]. However, none of these factors has been studied in a systematic fashion.

Simulation studies: Most simulation studies confirm that the characteristics of the dataset at hand have a strong influence on the relative performance of the various clustering methods. Therefore, one has to know the characteristics of the dataset at hand in order to choose a clustering method that is suitable for it. However, the data characteristics are generally unknown, so that *simulation studies offer little guidance in choosing a clustering method* [148, 150, 151]. Moreover, it has been observed that simulated datasets are somehow well-behaved, whereas real-world datasets are generally not

[55, 144]. This raises the question *whether simulation studies can possibly offer any insight into how clustering methods perform on real-world datasets.*

Analytic cluster validation: *(i)* Some theoretical results were observed to *contradict empirical evidence.* For example, the admissibility and the axiomatic approaches of Section 5.2.4 both rate the single-link method first. However, virtually all simulation studies show that the single-link method performs rather poorly even on error-free datasets [148, 150, 151]. *(ii)* Kaufman and Rousseeuw point out that it is *relatively easy to conceive a formal framework* that makes any arbitrary clustering method look best [117]. *(iii)* Analytic validation approaches *systematically ignore the effect of all kinds of heuristics,* which are so fundamental to most clustering methods [55].

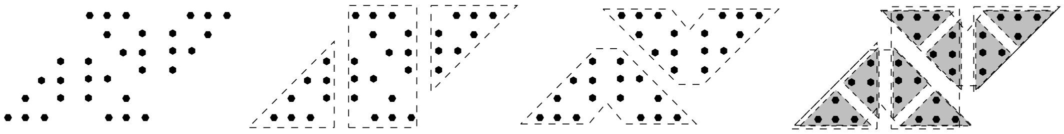
Other validation approaches, such as those described in Section 5.2.5, are mostly limited by their heuristic and ad hoc nature, which makes their objectivity questionable.

Aside from the above very concrete criticism, there is the more philosophical concern that cluster validity is an ill-defined concept. In fact, there is not even a generally accepted definition of what a cluster is. For example, partitioning methods (cf. Section 4.1.2) define a cluster as a group of objects that optimize a quality criterion. Density-based methods, by contrast, define a cluster as a dense region in data space. Hierarchical methods have yet another definition. So, relative to which definition should validity be measured?

Figure 5.2 illustrates this dilemma. Clearly, the cluster structures A and B are both meaningful, but fundamentally different. In fact, they reflect different aspects of the same dataset. Thus, the notion of a single "right" or "valid" cluster structure seems unnecessarily narrow [9]. Moreover, none of the two cluster structures A or B identifies the embedded sub-clusters shown in Figure 5.2(d). Does this mean that the structures A and B are invalid? The answer lies in the eye of the beholder.

5.3.2 Towards the Validation of Alarm Clusters

Based on the discussion of the previous section, we decided to use none of the classic validation techniques. In fact, external criteria are the only methods that establish a strong and truly objective sense of validity. All the other methods can only increase our comfort with a cluster structure, which is not fundamentally different from what manual validation can do. However, the problem with external criteria is that they need the "correct" cluster structure, which is hard to obtain in practice. One might try to obtain the correct cluster structure by manually



a) Sample dataset. b) Clustering A. c) Clustering B. d) Sub−clusters.

Figure 5.2: A sample dataset, two cluster structures, and their sub-clusters.

clustering alarm logs. However, manual clustering is error-prone, which makes it questionable whether the correct cluster structure can be identified in this way. Moreover, manual clustering does not scale, which is problematic because there is little value in showing that our alarm clustering method is correct for one, two, or five alarm logs. Alternatively, one could try to set up a controlled laboratory experiment, where all the root causes and their corresponding alarm clusters are known. However, the value of such an experiment is very limited unless the laboratory environment is representative of real environments, a requirement that is very hard to meet [144]. Moreover, it is unclear how many laboratory experiments are needed to validate the alarm clustering method.

To summarize, external criteria are very difficult to apply, but they are the only way to establish validity in a truly objective way. Given this situation, we approached the validation problem in two ways: First, we incorporated the idea of replication analysis into the alarm clustering method and thereby improved its robustness. Second, we developed two techniques that support the manual investigation of alarm clusters. Using these techniques, it becomes easier for the human analyst to identify and reject clusters of questionable quality.

Improving robustness: We like the idea of replication analysis according to which a cluster structure is more trustworthy if it is not affected by spurious or arbitrary decisions that were involved in its generation (cf. Section 5.2.2). However, summarizing the results of a replication analysis in a single number seems overly formal given that this number is interpreted in a qualitative way, where "larger is better". Moreover, the fourth step in a replication analysis partitions the second sample according to its resemblance to the clusters in the first sample. The details of this step are open to interpretation, and few guidelines exist. However, the choices made here are known to influence the final results [27]. Therefore, we modified the replication analysis of Section 5.2.2 in a way that addresses these concerns.

Let L be an alarm log and let $L_0 \subseteq L$ be a randomly chosen sub-log of L. We randomly partition the alarm log $L \setminus L_0$ into two equal-sized sub-logs L_1 and L_2

(i.e. $|L_1| = |L_2|$ and $L_1 \uplus L_2 = L \smallsetminus L_0$). Then, we apply the alarm clustering method separately to the logs $L_0 \cup L_1$ and $L_0 \cup L_2$ and compare the resulting generalized alarms. One could mark generalized alarms that are not discovered in both alarm logs as questionable, so that they are investigated more carefully. However, generalized alarms that do not replicate are very rare so that we decided to discard them without mention. The advantage of this approach is that it improves the robustness of the alarm clustering method while being completely transparent to the user. The user only sees "robust" alarm clusters.

Determining the size of the alarm log L_0 is an important decision. The larger L_0 is, the more alarms the two alarm logs $L_0 \cup L_1$ and $L_0 \cup L_2$ share, and the weaker the form of robustness that we enforce by means of replication. In the experiments of Chapter 6, we chose L_0 to contain 60% of the alarms in L. This value is based on informal experiments with different sizes of L_0.

Supporting the manual investigation of alarm clusters: After alarm clusters have been extracted, we rely on a human expert to investigate them and to decide whether they are valid or not. To support this manual validation process, we have implemented descriptive statistics and the technique of drill-down. *Descriptive statistics* make generalized alarms more informative by associating histograms with their attribute values. For example, consider a generalized alarm that models an alarm cluster of 5000 alarms, and assume that the source IP of this generalized alarm has the value "Web server". The histogram associated with the value "Web server" shows on the x-axis all Web servers that are subsumed by the concept "Web server". The y-axis of the histogram shows for each individual Web server how many out of the 5000 alarms it has triggered. In that way, it is possible to expose the elementary attribute values that are subsumed by a generalized one. In practice, we found that descriptive statistics significantly facilitate the understanding of generalized alarms.

Finally, *drill-down* is a well-known technique in data mining [90]. Drill-down allows the user of the alarm clustering method to reversely traverse the merge steps that lead to an alarm cluster. Hence, if an alarm cluster looks questionable, then the user can look at the alarm clusters that were merged to obtain it. If needed, drill-down can recursively continue down to the level of individual alarms. We found that drill-down offers a very practical method for investigating alarm clusters, and for establishing or refuting their validity.

5.3.3 Concluding Remarks

It is worth pointing out that our alarm clustering method has a control flow that is favorable for the creation of valid alarm clusters. In fact, beginning with the most apparent cluster, it repeatedly finds and removes clusters from the dataset. In doing so, the most clear-cut clusters are found and removed first, and the search for additional clusters continues with the left-over alarms. Anderberg recommends this strategy as one that is likely to find meaningful clusters [9]. Moreover, up to min_size alarms that did not fit well into any cluster might be left over at the end of clustering. This, it has been argued [92], is more natural than assigning all alarms (noise and outliers included) to clusters.

Chapter 6

Validation of Thesis Statement

This chapter is dedicated to the experimental validation of the thesis statement, which was given in Section 1.2. Section 6.1 presents a detailed example of how alarm clustering and root cause analysis work in practice. The remaining sections address the individual claims of the thesis statement. Specifically, Section 6.2 contains summary statistics from a large number of experiments to show that a few root causes account for the majority of alarms, and that these root causes can be discovered efficiently by means of alarm clustering. Section 6.3 shows that the future alarm load decreases quite significantly if root causes are removed or their associated alarms are filtered out. Section 6.4, finally, shows that filtering can be made safe so that there is a small risk of discarding true positives.

6.1 An Illustrative Example

This section illustrates how alarm clustering and root cause analysis work in practice. To this end, we consider the alarm log consisting of the 156380 alarms that IDS 7 of Table 1.2 triggered in April 2001. IDS 7 is deployed at a large commercial site whose network topology is sketched in Figure 4.3(a). To obtain a more accurate picture of the network topology, please replace the subnets *Net-A*, …, *Net-Z* of Figure 4.3(a) by the 2^{24} subnets 0.0.0.*, 0.0.1.*, …, 255.255.255.*, which are defined by the top three bytes of an IP address.

We configured the alarm clustering method with the following generalization hierarchies: For IP addresses and port numbers, we used the generalization hierarchies of Figure 4.3(c). As before, we ask you to mentally replace the subnets *Net-A*, …, *Net-Z* by the subnets 0.0.0.*, …, 255.255.255.*. For time-stamps, we employed the generalization hierarchies of Figure 4.4, with the time-stamp attribute being replicated as described in Section 4.2.3. For the context attribute, we chose to use generalization hierarchies based on frequent substrings, which is also described in Section 4.2.3. No generalization hierarchy was defined for the alarm type. Hence, the alarm type cannot be generalized and alarm clusters can never mix alarms of different alarm types. We found this to be a useful heuristic to obtain more understandable alarm clusters.

We consider the above generalization hierarchies to be simple and intuitive. Moreover, the experiments in Section 6.2 confirm that generalization hierarchies like these work generally well in practice. Finally, an important reason for choosing the aforementioned generalization hierarchy for IP addresses is that we were in possession of the necessary background knowledge to build it. For example, even if we had wanted to, we could not have used generalization hierarchies that structure IP addresses by their administrative domains; we simply did not have the necessary background knowledge. Thus, the availability of background knowledge affects the kind of generalization hierarchies that one can use in practice.

Table 6.1 shows the generalized alarms of the thirteen largest alarm clusters that we found. Each line of the table represents one alarm cluster and the "Size" column indicates the cluster's size. Throughout the table, "any" is generically written for attributes that have been generalized to the root of their generalization hierarchies. The value "undefined" in the "Context" column indicates that the IDS did not store any value for the context attribute. Similarly, the port attributes are occasionally undefined. For example, the ICMP protocol has no notion of ports [196]. As a consequence, the port attributes of "Fragmented ICMP Traffic" alarms are undefined. Finally, recall that the names *ip1*, *ip2*, ... refer to the machines in Figure 4.3.

Note that the generalized alarms of Table 6.1 summarize 95% of all alarms. We have therefore found a very crisp summary of almost the entire alarm log. Moreover, using this summary for root cause analysis is a huge simplification over using the original alarm log. Nevertheless, generalized alarms can only suggest root causes. Additional work is needed to validate them. This validation generally requires access to the alarm log, good security skills, and an understanding of the computing environment. Below we present the validated root causes that we found for the generalized alarms of Table 6.1. The discussion proceeds by alarm type:

WWW IIS View Source Attack: The first two generalized alarms of Table 6.1 contain the following (sanitized) substring in their context attributes:

GET /search_cgi/cgi?action=View&VdkVgwKey=
http%3A%2F%2Fwww**%2E**xyz**%2E**com

This request is completely legal and, based on Table 6.1, it has been issued more than 100000 times. Our analysis has shown that "WWW IIS View

Table 6.1: Generalized alarms of the thirteen largest alarm clusters.

Alarm Type	**Src-Port**	**Src-IP**	**Dst-Port**	**Dst-IP**	**Time-Stamp**	**Context**	**Size**
WWW IIS View Source Attack	Non-priv.	Internet	80	*ip1*	any	see text	54310
WWW IIS View Source Attack	Non-priv.	Internet	80	*ip2*	any	see text	54013
WWW IIS View Source Attack	Non-priv.	Firewall	80	Internet	any	any	17830
FTP SYST Command Attempt	Non-priv.	Firewall	21	Internet	any	any	6439
FTP SYST Command Attempt	Non-priv.	Internet	21	HTTP/FTP	any	any	4181
IP Fragment Attack	undefined	*ipA1*	undefined	*ip4*	Workday	undefined	4581
IP Fragment Attack	undefined	*ipA1*	undefined	*ip5*	Workday	undefined	3708
TCP SYN Host Sweep	Non-priv.	*ip4*	80	Internet	any	undefined	741
TCP SYN Host Sweep	Non-priv.	*ip5*	80	Internet	any	undefined	651
TCP SYN Host Sweep	Non-priv.	Firewall	25	Internet	any	undefined	250
Fragmented ICMP Traffic	undefined	Internet	undefined	*ip1*	any	undefined	823
Fragmented ICMP Traffic	undefined	Internet	undefined	*ip2*	any	undefined	711
Unknown Protocol Field in IP Packet	undefined	*ipZ1*	undefined	Firewall	End-of-month, Tuesday	undefined	861

Source Attack" alarms occur when a URL contains "%2E", as is the case for the above URL. The root cause lies in the search engine that the Web servers *ip1* and *ip2* offer. In fact, all the URLs that the search engine returns in response to client requests have their dots replaced by "%2E", which is the hex-encoding for a dot. When a client clicks on one of the returned search results, then the above alarm is triggered. Finally, the third generalized alarm in Table 6.1 turned out to be the reverse problem: Internal clients requesting external Web pages, the URL of which contains "%2E". Note that the internal clients are proxied by the firewall.

FTP SYST Command Attempt: These generalized alarms simply highlight the fact that many FTP clients issue the SYST command — a legal command that returns information about the FTP server. The root cause is the configuration of the FTP clients that tells them to issue the SYST command at the beginning of each FTP session.

IP Fragment Attack: Either *ipA1* is maliciously sending fragmented packets to the firewalls or there is a router that fragments the packets between *ipA1* and the firewalls. Our investigation has convinced us that the fragmenting router is the actual root cause.

TCP SYN Host Sweep: Here, the IDS thinks that the firewalls are running host sweeps. In reality, however, the firewalls proxy the HTTP (port 80) and SMTP (port 25) requests of their clients. While exercising this function, the firewalls occasionally contact many external machines at virtually the same time. The resulting traffic resembles host sweeps.

Fragmented ICMP Traffic: After investigating the source IPs, we realized that they all belong to various Internet Service Providers (ISPs). Therefore, we conjectured that there is some link between fragmented ICMP traffic and modem access to the Internet, i.e. the root cause is some particularity of certain modems.

Unknown Protocol Field in IP Packet: At the end of the month, a machine on the Internet starts using an unknown transport layer protocol to communicate with the firewall. As many security tools ignore protocols that they do not understand, attackers occasionally use unknown protocols to establish covert channels. A closer investigation of this generalized alarm seems to indicate that, indeed, *ipZ1* is using a covert channel.

Note that for the last two alarm types, we had some difficulty in pinpointing the actual root causes. This, however, is no limitation of our alarm clustering method. In fact, even when we looked at the raw intrusion detection alarms, we could not ascertain the root causes. Too much information was missing. For example, what modems do the ISP clients use, and what hardware do the ISPs deploy? Who is *ipZ1*, and what is the "unknown protocol" that has been observed? These questions are hard to answer, partially because the IDS provides too little information about the "unknown protocol", partially because certain components such as the modems or the machine attached to *ipZ1* are out of our control. Hence, with or without alarm clustering, there are cases where we do not have enough information to identify root causes with certainty.

Recall that we are experimenting with historical (N.B. real-world) alarm logs. Therefore, we cannot simply remove the previously identified root causes. As an alternative, we used filtering to estimate the alarm load reduction that we could expect. Specifically, we wrote filtering rules that discarded all alarms matching one of the generalized alarms in Table 6.1. We then applied these filters to the alarms of the following month. The result was that 82 percent of all alarms in May were automatically discarded by the filtering rules. Thus, if the root causes had been resolved, then 82 percent less work would have been the estimated payoff in May 2001. Section 6.4 discusses the risk that filtering discards true positives.

6.2 Experience with Alarm Clustering

The first and second claim of the thesis statement say that a few root causes generally account for the majority of alarms, and that these root causes can be discovered efficiently by means of alarm clustering (cf. Section 1.2). This section contains summary statistics from a large number of experiments to substantiate these claims. Section 6.2.1 describes the experimental setup, and Section 6.2.2 presents the results obtained.

6.2.1 Experimental Setup

The experimental setup is very similar to the one of the last section. Specifically, for each IDS of Table 1.2 and each month of the year 2001, we take all alarms of this IDS and month, and store them into a separate alarm log. This yields $(16\,\text{IDSs}) \times (12\text{ months}) = 192$ pairwise disjunct alarm logs. We run the alarm clustering method individually on each of these alarm logs and record the results.

As above, we use the generalization hierarchies of Figure 4.3(c), Figure 4.4, and Section 4.2.3 for port numbers, the time-stamp, and the context attribute, respectively. No generalization hierarchy is defined for the alarm type. For IP addresses we adapt the generalization hierarchy of Figure 4.3(c) to the specific environments of the different IDSs. This is done as follows: The root concept of the adapted hierarchy is called *Any-IP* and represents the set of all possible IP addresses. The concept "Any-IP" is split into the categories *Protected IPs* and

External IPs. The protected IPs, which correspond to the DMZ of Figure 4.3(c), are those IPs that the IDS-operator is responsible for. External IPs are all the other IPs that belong to someone else. We further sub-classify the protected IPs according to the services (e.g. HTTP, FTP, Telnet, etc.) they offer. External IPs are further classified by subnet.

The alarm clustering method is configured to enforce ε-robustness (cf. Section 4.3.4) with $\varepsilon = 0.05$. Moreover, as described in Section 5.3.2, the alarm clustering method performs replication analysis, and only returns alarm clusters that are robust in the sense that they are independently discovered in two different sample logs that overlap in 60% of their alarms. Instead of manually setting the *min_size* values as required in the lines 2 and 9 of Figure 4.7, we mechanically set *min_size* to one tenths of the number of alarms in table T (i.e. $min\_size := |T|/10$). We run the alarm clustering method for all 192 alarm logs during the night, and manually review the results. The purpose of this review is not to actually identify the root causes, but rather to convince ourselves that the clustering results are informative and interpretable. When we find generalized alarms that are overly general or hard to interpret, then we repeat the alarm clustering process, but this time with manually set min_size values. The alarm clusters that we find this way are substituted for the mechanically generated ones. This yields the final set of generalized alarms, to which the results of the next section refer.

Two remarks are in order. First, we found that the default value for min_size works well in most cases, but occasionally, it is important to manually tweak the min_size value to the alarm log at hand. However, manually reviewing and revising the mechanically generated clustering results is very time consuming, which is why we had not done it in our earlier work [114]. In hindsight, we observed that this extra effort has improved the quality of the results reported here. The second remark concerns the replicability of the experiments reported here. The alarms used in our experiments originate from Fortune 500 companies, and — for confidentiality reasons — cannot be made public. However, the thesis statement of this dissertation concern intrusion detection alarms in general, not the specific alarms of our experiments. Therefore, anybody in possession of intrusion detection alarms should be able to verify the results reported here. Moreover, because of the limitations of simulated or laboratory environments [144], we consider it essential to use real-world alarms for the evaluation.

6.2.2 Alarm Clustering Results

The purpose of this section is to validate the first two claims of the thesis statement. This is done by means of summary statistics that we derived from an analysis of the previously mentioned 192 alarm logs. We first address the second claim of the thesis statement, according to which alarm clustering is a practical method for identifying root causes. One aspect of practicability is the run-time of the clustering method. The histogram of Figure 6.1 is based on time measurements taken on a 700 MHz Pentium III CPU with 1 GB RAM. All run-times include the time for reading and processing an alarm log. For example, Figure 6.1 shows that 24 out of the 192 alarm logs were analyzed in less than one minute. For 60 alarm logs, the analysis took between one and two minutes. Run-times of more than nine minutes were observed for only 22 alarm logs. Figure 6.2 shows that the average run-time is approximately linear in the log size. The peak for logs between 280000 and 320000 alarms is caused by an exceptionally large number of iterations in process that implements ε-robustness (cf. Section 4.3.4). In general, however, run-times are relatively low.

Another aspect of practicability is the number and interpretability of generalized alarms. In fact, if the alarm clustering method generated an excessive number of generalized alarms, or if generalized alarms were hard to interpret, then alarm clustering would be of minor help for root cause analysis (contrary to the second claim of the thesis statement). Figure 6.3 shows the average number of generalized alarms per IDS, with the averages being taken over all months of the year 2001. It becomes apparent that on the average over all 192 alarm logs, there are 18 generalized alarms per alarm log. It is clearly practicable to perform a manual root cause analysis for an average of 18 generalized alarms per alarm log. As a matter of fact, for 48 of the 192 alarm logs we have used the respective generalized alarms to perform the root cause analysis (see Section 6.3 for the selection of alarm logs.) For the sake of objectivity, we asked colleagues to perform independently from us the root cause analysis for 10 of the 48 alarm logs. Their and our results were consistent. Moreover, we and our colleagues agreed that generalized alarms were intuitive and easy to interpret. Finally, interpreting the generalized alarms took us between one and two hours per alarm log. Taken together, these findings support the second claim of the thesis statement.

By way of illustration, we next list some of the more interesting root causes that have not yet been reported in this dissertation:

- In one Windows network, we observed a large number of "Anonymous session" alarms. In the present case, this was part of the normal system operation. IDSs report anonymous sessions, because an attacker can use them to enumerate users, shares, and under Windows NT even registry information.
- Between April and September 2001, the number of DNS-related scans and attacks increased one-hundred-fold on most IDSs that monitored Internet traffic. We conjecture that this spike is related to a January 2001 CERT advisory that warned of multiple vulnerabilities in BIND, which is a widely used implementation of the DNS service [32].

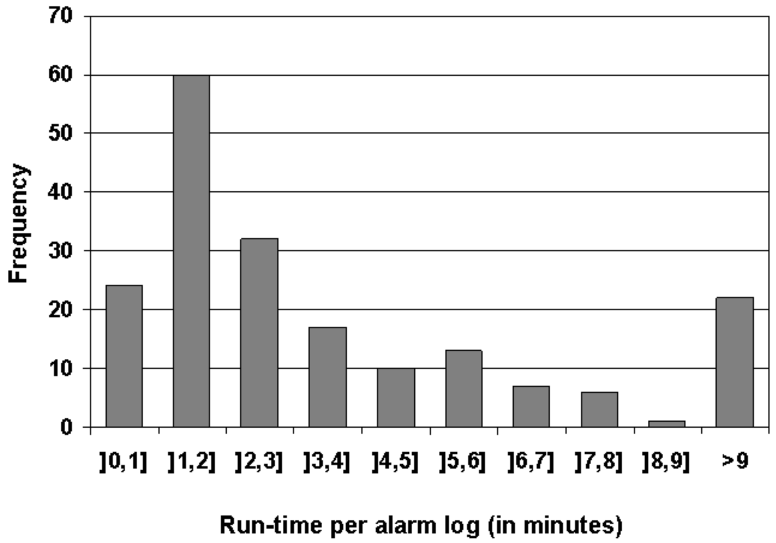


Figure 6.1: Histogram showing the frequencies of different run-times.

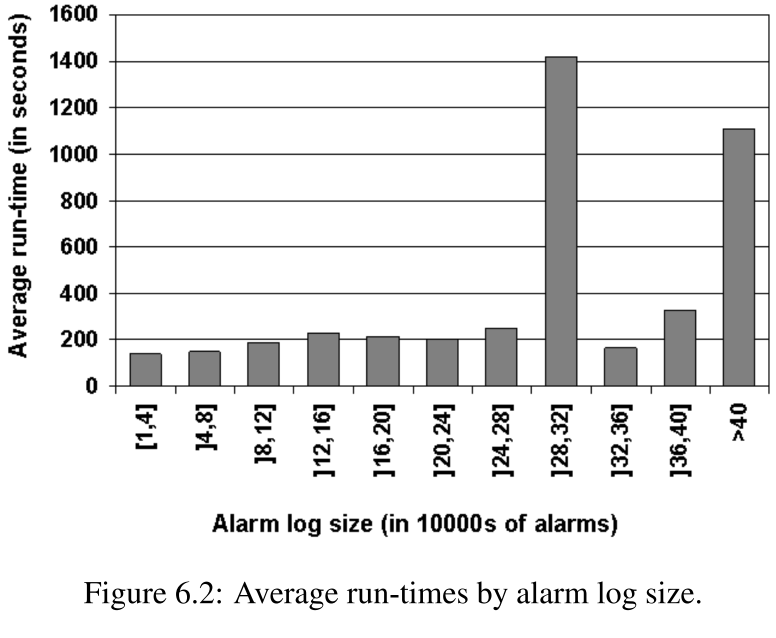


Figure 6.2: Average run-times by alarm log size.

- A broken router triggered an abundance of “TCP overlap” alarms. IDSs report overlapping TCP traffic because it has been used to evade attack detection [176].
- Some IP addresses systematically scanned the default ports of well-known Trojan horses.
- Consider a DNS client that sends a batch of DNS requests to a DNS server. When the DNS server returns its replies, it returns a series of UDP packets to various ports of the DNS client. This traffic pattern looks as if the DNS server was UDP port-scanning the DNS client, which triggers alarms on most IDSs.
- Some IDSs trigger on URLs containing “/exec/” because this substring is associated to a vulnerability in the Cisco Catalyst 3500 XL switch (cf. CVE-2000-0945). The users at one site were eager on shopping at amazon.com. However, virtually all URLs at amazon.com contain the substring “/exec/”, which resulted in a correspondingly large number of false positives.
- Some sites had large amounts of Napster traffic, which some IDSs report because companies generally prohibit the use of Napster, and similar file-swapping applications.
- Socksified FTP clients always trigger an abundance of “FTP bounce” alarms. In detail, FTP clients can instruct FTP servers to send their data to any IP address and port of the client’s choice. This ability can be abused for the FTP bounce attack [31], in which the FTP client specifies an IP address different from its own. However, socksified FTP clients are proxied by a SOCKS server [133], and have to specify the IP address of the SOCKS server, rather than their own. This triggers “FTP bounce” alarms.

We now turn to the first claim of the thesis statement, according to which a few root causes generally account for the majority of alarms. This claim is confirmed by our experience that a few predominant and persistent root causes systematically trigger large amounts of mostly false alarms. Chapter 3 has made this point in great detail, and it is further supported by the above sample root causes. To defend this claim in a quantitative manner, we have to identify a small number of root causes, and show that these root causes triggered the majority of alarms. Using the alarm clustering method, it is relatively easy to identify root causes, but it is very difficult to count the number of alarms that these root causes have triggered. In fact, Section 3.2 has shown that it is generally not possible to decide with certainty whether a given root cause has triggered a particular alarm. Hence, it is equally impossible (at least in the general case) to count the number of alarms that a root cause has triggered. For these reasons, we opted to approximate the number of alarms per root cause.

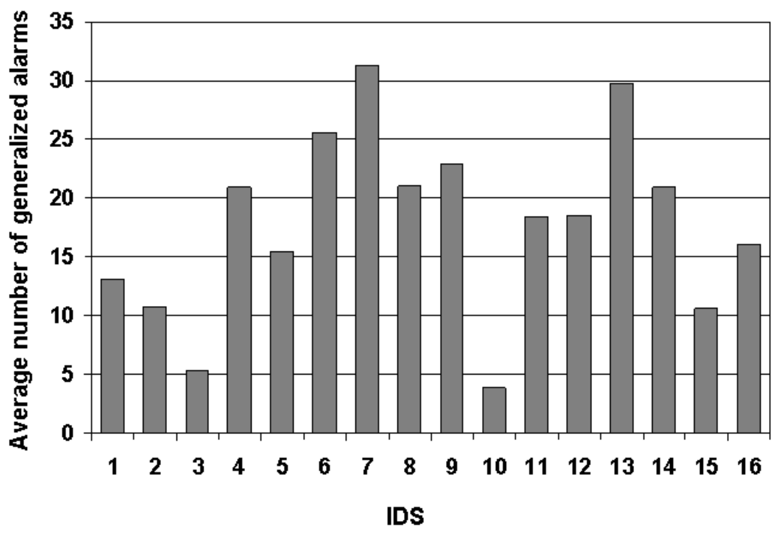


Figure 6.3: Average number of generalized alarms per IDS.

For the purpose of this approximation, we assume that our alarm clustering method is correct in the sense of Definition 3.2. In other words, let L be an alarm log, and let $C_1, \ldots, C_n$ be the alarm clusters that the alarm clustering method has found for L. Then, we assume that all alarms in cluster C_i share the same root cause, $i = 1, \ldots, n$. Two conclusions are immediate:

1. Because each alarm cluster corresponds to a root cause, we have established the existence of $k \leq n$ root causes, with $k = n$ if the alarm clusters have pairwise distinct root causes. Note that the min_size parameter controls the number n of alarm clusters, and hence, the number k of root causes. Clearly, this control is weakened by the effect that ε-robustness (cf. Section 4.3.4) has on the clustering results. Nonetheless, it might seem that the above correctness assumption (from which we concluded the existence of $k \leq n$ root causes) is a strong one.
2. These k root causes account for $(100/|L|) \times \sum_{i=1}^{n} |C_i|$ percent of the alarms in L. This percentage is called the *coverage*.

To show that a few root causes account for the majority of alarms, we next show that k is generally small in our experiments, while the coverage is large. In fact, Figure 6.3 shows that the number n of alarm clusters per alarm log is relatively small, namely 18 on the average. Consequently, the number k of root causes is small, as well. The histogram of Figure 6.4 shows the number of times the different degrees of coverage are obtained in our experiments. For example, between 95% and 100% of coverage is obtained for 59 of the 192 alarm logs. For 51 alarm logs, the coverage is between 90% and 95%, and on the average,

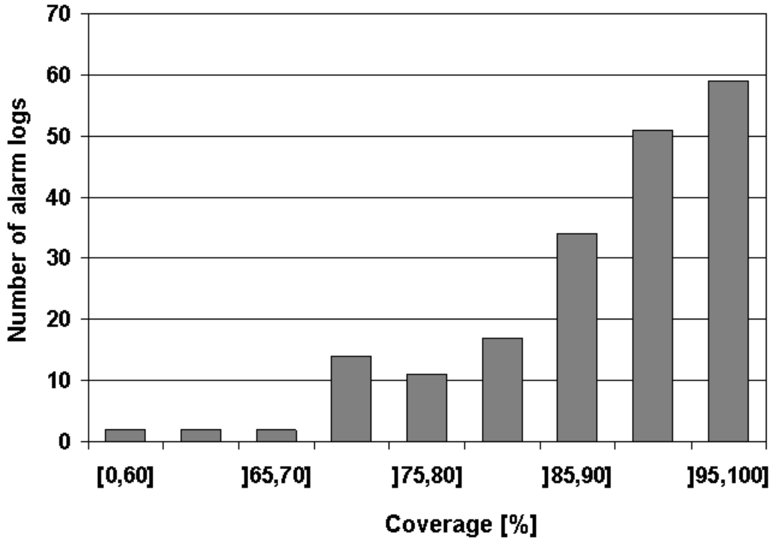


Figure 6.4: Histogram showing the frequency of different degrees of coverage.

an alarm log has 90% coverage. Hence, an average of less than 18 root causes typically accounts for 90% of the alarms in an alarm log. This confirms the first claim of the thesis statement.

6.3 Alarm Load Reduction

According to the third claim of the thesis statement, one can significantly reduce the future alarm load by removing root causes, or by filtering out the alarms that originate from benign root causes. To verify this claim, one has to measure the effect of root cause removal and alarm filtering. However, given that we are experimenting with historical (N.B. real-world) alarm logs, it is not possible to actually remove root causes. Therefore, we use only filtering in our experiments. Note that the use of filtering is born of necessity, and does not endorse filtering over root cause removal. In a nutshell, we choose a month m in the year 2001 and an IDS I from Table 1.2. Then, we perform a root cause analysis for the alarm log corresponding to IDS I and month m. Based on the results obtained, we derive filtering rules and apply them to the alarms that IDS I triggers in month $m+1$. In doing so, we calculate the percentage of alarms that are discarded by the rules. The higher this percentage, the stronger the evidence in favor of the thesis statement.

To perform a root cause analysis for a given IDS, one needs a detailed understanding of the site at which the IDS is deployed. Gaining such an understanding is time-consuming and requires the close cooperation of the IDS owner. For lack of time and resources we were not able to gain a sufficiently deep understanding for all IDSs. Therefore, we decided to focus on the IDSs 3, 6, 10, and 14, which are representative in the following ways: First, the IDSs 3 and 6 are of type A, while the IDSs 10 and 14 are of type B, where A and B are pseudonyms for the IDS vendors (cf. Section 1.4). Second, IDS 3 is located in a DMZ, IDS 6 is on an extranet, IDS 10 is on an intranet, and IDS 14 is on a link to the Internet. Third, according to Table 3.2 on page 48, IDS 3 seems to contradict the alarm cluster hypothesis, IDS 14 offers some evidence in favor of the alarm cluster hypothesis, and the IDSs 6 and 10 offer strong evidence in favor of the alarm cluster hypothesis. Finally, for each of the IDSs 3, 6, 10, and 14 we consider all months of the year 2001, i.e. $m = \text{Jan}, \ldots, \text{Dec}$. This results in the 4 IDSs $\times$ 12 months $= 48$ alarm logs mentioned in the previous section.

After completing the root cause analysis for IDS I, $I \in \{3, 6, 10, 14\}$, and month m, $m \in \{\text{Jan}, \ldots, \text{Dec}\}$, we wrote filtering rules that discarded all alarms belonging to the identified root causes. In most cases, the filtering rules were identical to the generalized alarms. Occasionally, we adapted the generalized alarms according to our understanding of the root causes to obtain more adequate filtering rules. Because root cause removal was no option, we wrote filtering rules for benign and malicious root causes alike. In the real world, one might choose to filter out true positives if their targets are not vulnerable [136], but other responses such as firewall reconfigurations are more typical. Either way, the end effect is that in the future, one no longer has to manually handle the alarms of well-understood root causes. In our experiments, we imitate this automation by means of filtering rules. We applied the derived filtering rules to the alarms that the same IDS triggered in the following month. We measured the *alarm load reduction*, i.e. the percentage of alarms that were discarded by the filtering rules, and plotted it in the Figures 6.5 through 6.8.

For example, Figure 6.5 shows that the filtering rules that we derived from the January alarms of IDS 3 discard 93% of the alarms that IDS 3 triggers in February 2001. In other words, the number of alarms that the IDS operator had to investigate in February could have been reduced by 93% if root cause analysis in combination with filtering had been done at the end of January. Note that the alarm load reduction for January 2001 is not available, because we do not use the alarms from December 2000 in our experiments. Further, the rules derived from the February alarms of IDS 3 discard 92% of the March alarms, and so on. It becomes evident from the Figures 6.5 and 6.7 that the IDSs 3 and 10 have a relatively high alarm load reduction of approximately 95% on the average. On the other hand side, the alarm load reduction of the IDSs 6 and 14 is much more modest. This variability can be explained as follows:

IDS 3 is deployed in a Web hosting environment. In this environment, the bulk of alarms are TCP SYN host-sweeps, which result from external hosts

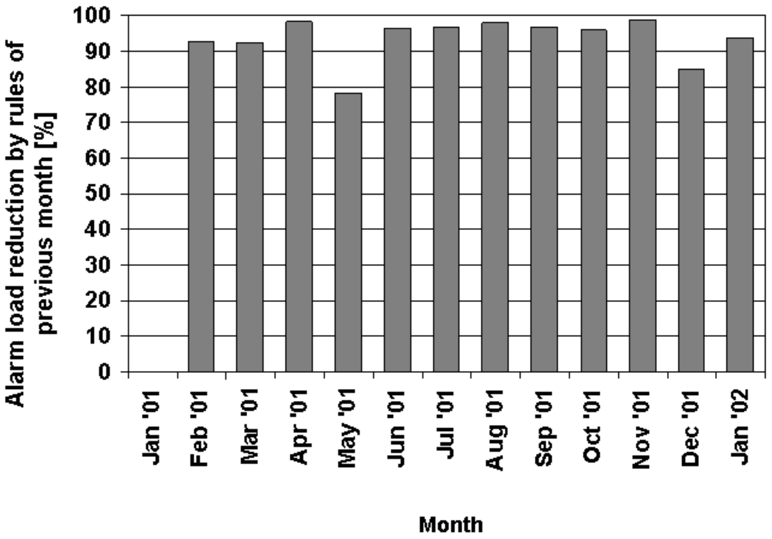


Figure 6.5: Alarm load reduction for IDS 3.

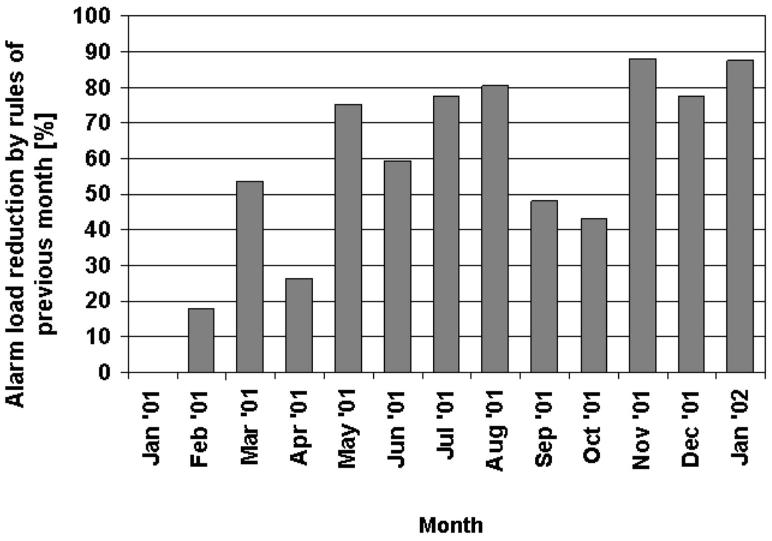


Figure 6.6: Alarm load reduction for IDS 6.

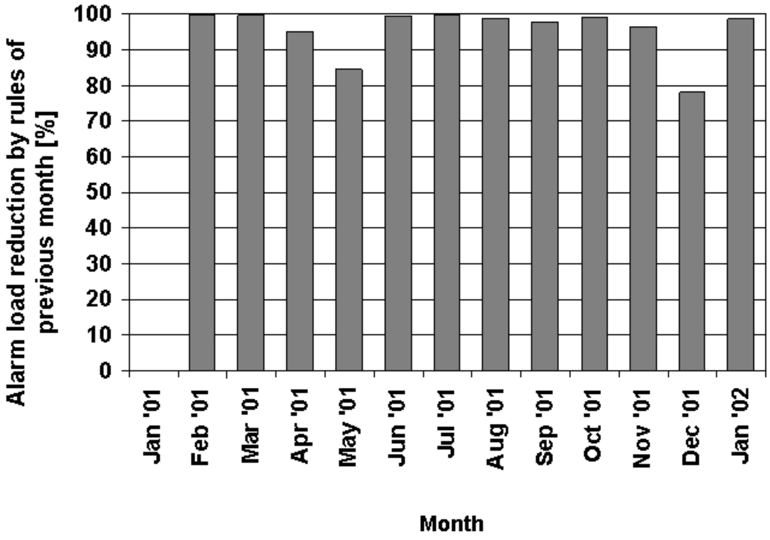


Figure 6.7: Alarm load reduction for IDS 10.

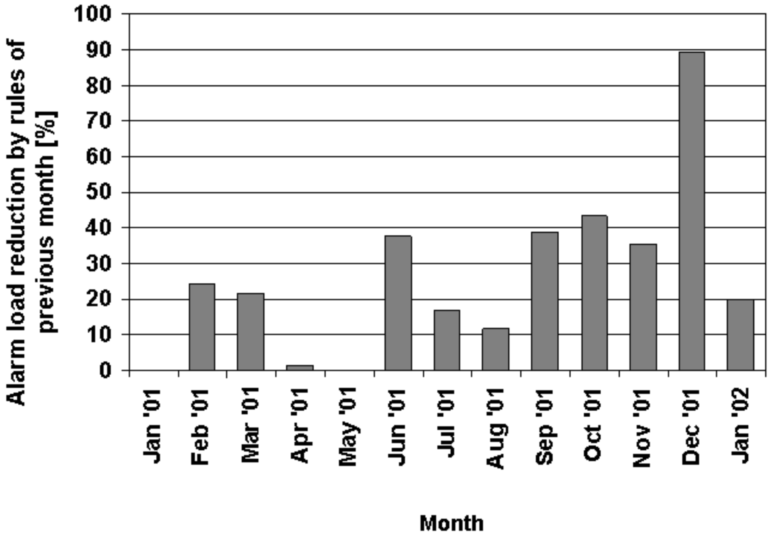


Figure 6.8: Alarm load reduction for IDS 14.

contacting in short succession a series of distinct Web servers. The resulting scan-like traffic patterns are extremely repetitive and predictable. This makes filtering very effective and results in an average alarm load reduction of approximately 95%.

IDS 6 features an over-proportionally low alarm load reduction in February, April, September, and October. Based on our investigation, a temporary networking problem on the 5th and 6th of February has triggered 563205 ”Fragmented IP” alarms. Clearly, the filtering rules derived from the January alarms cannot anticipate this temporary networking problem. As a consequence, they are ineffective, and the alarm load reduction in February drops to 18%. In April, the already mentioned spike in the number of DNS scans and attacks renders the filtering rules from March ineffective. The advent of NIMDA in mid-September, as well as other unpredictable attacks explain the alarm load reduction of only 48% in September. By mid-October, NIMDA has spread to the point where the filtering rules from September are too specific to be effective. Hence the alarm load reduction of 43% in October.

IDS 10 is deployed on an internal network. Our investigation has shown that the vast majority of alarms result from a network management system. Moreover, these alarms are extremely repetitive, so that root cause analysis followed by filtering achieves the very high alarm load reductions shown in Figure 6.7.

IDS 14 is deployed before the firewall of a large company. Based on our analysis, the majority of alarms triggered by IDS 14 report real attacks. However, most real attacks are ephemeral, and do not progress for months in a row. This explains why the filtering rules derived in one month yield a generally low alarm load reduction in the following month. The 90% alarm load reduction in December is an exception, which is due to a significant amount of port scanning during the months of November and December. The alarm load reductions of nearly 0% in April and May are due to a temporary networking problem that triggers 1415167 ”TCP Overlap Data” alarms in April. Clearly, the filtering rules from March cannot anticipate this networking problem, and are consequently ineffective in April. On the other hand side, the filtering rules from April deal mostly with ”TCP Overlap Data” alarms, which do no longer occur in May. Hence the poor alarm load reduction in May.

The average alarm load reduction over the 48 months of the Figures 6.5 through 6.8 is 70%. Even though this percentage is relatively high, we believe that it actually underestimates the attainable alarm load reduction. In fact, the above examples show that the sudden advent of a new root cause (e.g. a networking problem or a worm) can trigger a sharp drop in the alarm load reduction. After all, the filtering rules were derived from alarms that contained no trace of these new root causes. Consequently, the rules are ineffective at handling the new alarms, and the alarm load reduction drops. This situation is unaltered when root cause removal is used instead of filtering, because one cannot remove root causes that do not yet exist. However, one can attempt to detect and respond as fast as possible to new root causes. To this end, one can monitor the average number of alarms per time unit. When this number rises sharply, it is a strong indication that new root causes have set in. Then, one can immediately re-do the root cause analysis and respond to the root causes discovered. This approach is clearly superior to our high-latency approach, where root cause analysis is done once a month. Moreover, the new approach is likely to result in an average alarm load reduction that is even higher than the 70% reported above.

While running the above experiments, we observed that the alarm clustering method tended to produce generalized alarms that — according to our root cause analysis — could be used as filtering rules. We therefore hypothesize that generalized alarms are generally good filtering rules. If one is willing to accept this hypothesis, then it is not necessary to actually perform a root cause analysis to derive filtering rules. Rather, one can simply use the generalized alarms as *default filtering rules*. Using this idea, we have calculated the average alarm load reductions for all 16 IDSs of Table 1.2. Figure 6.9 shows the resulting averages. For example, in the year 2001, the default filtering rules for IDS 1 achieve an average alarm load reduction of 90% per month. The average alarm load reduction for IDS 2 is 82%. Not surprisingly, the averages for the IDSs 3, 6, 10, and 14 are very close to the average alarm load reductions of the manually derived filtering rules. These latter averages can be calculated from the Figures 6.5 through 6.8. The average alarm load reduction in Figure 6.9 is 70% per IDS and month. This is consistent with the claimed effectiveness of filtering.

6.4 On the Risks of Filtering

The thesis statement of Section 1.2 claims that it is safe to filter out alarms that have benign root causes, where *safe* means that the risk of discarding true positives is small. This section defends this claim. Doing so is no easy endeavor because there are many factors that influence how safe the use of filtering rules actually is. Three particularly noteworthy factors are:

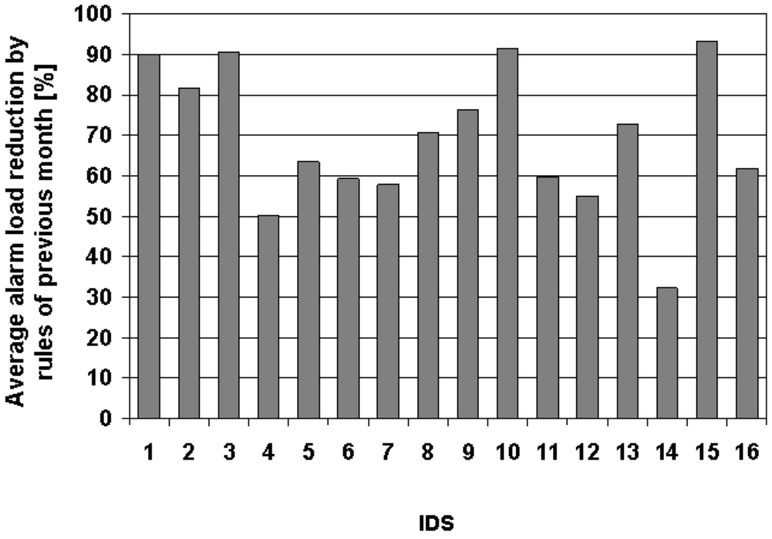


Figure 6.9: Average alarm load reduction per IDS.

The author: Recall that filtering rules are written based on one's understanding of alarm root causes. Hence, whether a filtering rule is safe or not depends to a good part on the author who wrote it.

The attacker: The safety of filtering rules is also contingent on how knowledgeable attackers are. In fact, knowledge of filtering rules makes it easier for an attacker to design an attack so that the resulting alarms match a filtering rule and get discarded.

The environment: In a very dynamic environment there is a higher risk that a formerly safe filtering rule becomes obsolete and starts discarding true positives. Conversely, the more static and the more tightly controlled an environment is, the safer it is to use filtering.

In summary, if the author is skilled, the environment does not change in ways that the author could not anticipate, and the attacker does not know the filtering rules, then filtering is safe. Clearly, to quantify safety in an objective manner, one first has to quantify these three factors. However, it is not clear how to quantitatively measure the author's skills, the stability of the environment, and the attacker's knowledge. This suggests that there probably is no simple solution to the problem of quantifying safety.

Moreover, we believe that a quantitative statement such as "any given true positive has a probability of 10^{-3} to be filtered out" is meaningless because safety must be seen in comparison to alternative ways of handling alarms. For example, if a part-time employee had a probability of 10^{-6} to miss a true positive, then filtering would be rather unsafe, given the above (hypothetical) probability of 10^{-3}. But what if it took 10 security specialists using correlation systems such as those of Section 2.4 to miss as few as one out of 10^6 true positives? At first sight, this seems to imply that filtering is comparatively unsafe. However, 10 well-equipped security experts are much more expensive than root cause analysis and filtering. It is therefore unfair to compare said security experts to our filtering technology. This raises another problem, namely the problem of deciding what to compare filtering to. At this point, it should have become evident that quantifying safety is a difficult research problem, which exceeds the scope of this dissertation. As an alternative to a quantitative analysis, the next section proposes four pragmatic ways to make filtering safer.

6.4.1 Guidelines for Safe Filtering

Users concerned about the safety of filtering are advised to abide by the following guidelines:

Write specific rules: The more specific a filtering rule is, the less likely it is to discard true positives. Ideally, filtering rules should always inspect the context attribute, which — as explained in Section 1.4 — stores the raw audit data that the IDS believes to contain an attack. In that way, filtering rules can double-check the analysis of the IDS and thereby guarantee that only false positives are discarded. Unfortunately, not all IDSs set the context attribute in all alarms.

Keep rules secret: Keeping filtering rules secret makes it more difficult for an attacker to "hijack" them.

Remove outdated rules: Computing environments are dynamic. Hosts, networks, and services come and go, and IDS software gets updated or even replaced. As a consequence, alarms that were predominant in the past may vanish, which renders their associated filtering rules obsolete. Obsolete filtering rules should be removed because they do not noticeably reduce the alarm load, while still bearing the danger to discard true positives.

Filter when not vulnerable: This recommendation does not make filtering safer, but it limits the harm done when a true positive is actually discarded. The idea is to filter alarms only when they report attacks that the target is not vulnerable to. In fact, some authors recommend to systematically filter out all alarms that affect non-vulnerable targets [136].

Chapter 7

Summary and Outlook

This chapter summarizes the work presented in this dissertation and reviews the main contributions made. Possible directions for future research are suggested.

7.1 Summary and Conclusions

This thesis proposes a new solution to the problem that intrusion detection systems overload their human operators by triggering thousands of mostly false alarms per day. Central to our solution is the notion of a root cause, which is the reason for which an alarm was triggered. The foundation of our solution is laid by three key contributions:

1. It is shown that, in general, a small number of root causes is responsible for the majority of alarms that an IDS triggers.
2. A new data mining technique is developed, and it is shown that applying this data mining technique to alarm logs constitutes an efficient approach for identifying root causes.
3. It is shown that once root causes have been identified, one can safely and significantly reduce the future alarm load by fixing them or by filtering out alarms that are associated to benign root causes.

Based on these contributions, we have introduced the following new alarm handling paradigm: Firstly, the proposed new data mining technique is used to identify root causes that account for large numbers of alarms, and secondly, these root causes are fixed or their associated alarms are judiciously filtered out (provided they are false positives). We have validated this alarm handling paradigm in extensive experiments with real-world intrusion detection alarms. On the average, we found that a few hours of time spent on the identification and removal of root causes would pay off by reducing the future alarm load by 70%.

Further contributions of this dissertation include a description of the link between data mining and root cause analysis (cf. Section 3.2), a high-level characterization of intrusion detection alarms (cf. Proposition 3.1), the development of heuristic rules for attack detection (cf. Section 3.5.2), and a case study for the application of data mining techniques to a real-world problem. Three conclusions can be drawn when comparing this work to previous work in alarm handling:

Console versus source: Most previous work has focused on building monitoring consoles that handle alarms as effectively as possible. This dissertation has shown that many alarms originate from persistent root causes. Further, it has been demonstrated that these alarms are best handled at their source (i.e. by eliminating their root causes) rather than at the console.

Online versus off-line: Most previous work has focused on online systems that analyze alarms in real-time. This dissertation has shown that there is substantial value in the off-line analysis of historical alarms.

True positives versus false positives: While most previous work has attempted to single out and escalate true positives, our work has shown the feasibility and usefulness of eliminating false positives. (Note that our alarm handling paradigm mainly eliminates false positives because most attacks are too ephemeral to be properly addressed by a high-latency approach like ours, which involves off-line data mining followed by some response.)

7.2 Future Work

This thesis work can be further pursued in one or more of the following directions:

- At present, interpreting alarm clusters in terms of root causes is still manual. However, one could envision to build an expert-system that automates the interpretation of alarm clusters. In this way, root cause analysis would become fully automatic.
- As discussed in Section 4.4, future work could investigate the integration of further forms of background knowledge into the alarm clustering method.
- Future work could continue the work of Section 6.4 and develop methods to quantitatively assess the risk of filtering.
- One could build an IDS that supports root cause analysis. At a minimum, such an IDS should have public (rather than proprietary) signatures and a verbose-mode, in which the traffic matching certain signatures is logged.

Bibliography

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 94–105, 1998.

[2] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling Multidimensional Databases. Technical report, IBM Almaden Research Center, 1995.

[3] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining Associations between Sets of Items in Massive Databases. In *Proceedings of the ACM-SIGMOD 1993 International Conference on Management of Data*, pages 207–216, 1993.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 487–499, September 1994.

[5] Mark S. Aldenderfer and Roger K. Blashfield. *Cluster Analysis*. Quantitative Applications in the Social Sciences. Sage Publications, 1984.

[6] Julia Allen, Alan Christie, William Fithen, John McHugh, Jed Pickel, and Ed Stoner. State of the Practice of Intrusion Detection Technologies. Technical report, Carnegie Mellon University, January 2000. `http://www.cert.org/archive/pdf/99tr028.pdf`.

[7] Magnus Almgren, Hervé Debar, and Marc Dacier. A Lightweight Tool for Detecting Web Server Attacks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2000)*, pages 157–170, February 2000.

[8] Magnus Almgren and Ulf Lindqvist. Application-Integrated Data Collection for Security Monitoring. In *Proceedings of the 4th Workshop on Recent*

Advances in Intrusion Detection (RAID), LNCS, pages 22–36. Springer Verlag, 2001.

[9] Michael R. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973.

[10] Phipps Arabie and J. Douglas Carroll. MAPCLUS: A Mathematical Programming Approach to Fitting the ADCLUS Model. *Psychometrika*, 45:211–235, 1980.

[11] Phipps Arabie and Lawrence J. Hubert. An Overview of Combinatorial Data Analysis. In Arabie et al. [12], pages 5–63.

[12] Phipps Arabie, Lawrence J. Hubert, and Geert De Soete, editors. *Clustering and Classification*. World Scientific Publishing, 1996.

[13] Stefan Axelsson. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[14] Rebecca Bace. *Intrusion Detection*. Macmillan Technical Publishing, 2000.

[15] G.H. Ball and D.J. Hall. ISODATA, a Novel Method of Data Analysis and Pattern Classification. Technical report, Stanford Research Institute, Menlo Park, CA, 1965.

[16] Jeffrey D. Banfield and Adrian E. Raftery. Model-Based Gaussian and Non-Gaussian Clustering. *Biometrics*, 49:803–821, 1993.

[17] Daniel Barbará and Sushil Jajodia, editors. *Applications of Data Mining in Computer Security*. Kluwer Academic Publisher, Boston, 2002.

[18] Daniel Barbará, Ningning Wu, and Sushil Jajodia. Detecting Novel Network Intrusions Using Bayes Estimators. In *Proceedings of the 1st SIAM International Conference on Data Mining (SDM'01)*, 2001.

[19] Steven M. Bellovin. Packets Found on an Internet. *Computer Communications Review*, 23(3):26–31, 1993.

[20] Michael J. A. Berry and Gordon Linoff. *Data Mining Techniques*. John Wiley and Sons, Inc., 1997.

[21] Gilles Bisson. Conceptual Clustering in a First Order Logic Representation. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 458–462, 1992.

[22] Eric Bloedorn, Bill Hill, Alan Christiansen, Clem Skorupka, Lisa Talboot, and Jonathan Tivel. Data Mining for Improving Intrusion Detection, 2000. `http://www.mitre.org/support/papers/tech_papers99_00/`.

[23] Hans-Hermann Bock. Probability Models and Hypotheses Testing in Partitioning Cluster Analysis. In Arabie et al. [12], pages 377–453.

[24] Anastasios T. Bouloutas, Seraphin B. Calo, and Allan J. Finkel. Alarm Correlation and Fault Identification in Communication Networks. *IEEE Transactions on Communications*, 42(2/3/4):523–533, 1994.

[25] Paul S. Bradley and Usama M. Fayyad. Refining Initial Points for K-Means Clustering. In *Proceedings of the 15th International Conference on Machine Learning*, pages 91–99, 1998.

[26] Paul S. Bradley, Usama M. Fayyad, and Cory A. Reina. Scaling EM (Expectation-Maximization) Clustering to Large Databases. Technical Report MSR-TR-98-35, Microsoft Research, 1998.

[27] James N. Breckenridge. Replication Cluster Analysis: Method, Consistency, and Validity. *Multivariate Behavioral Research*, 24(2):147–161, 1989.

[28] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 255–264, 1997.

[29] John Broderick – Editor. IBM Outsourced Solution, 1998. `http://www.infoworld.com/cgi-bin/displayTC.pl?/980504sb3-ibm.htm`.

[30] CERT. Advisory CA-1996-26: Denial-of-Service Attack via ping, 1996. `http://www.cert.org/advisories/CA-1996-26.html`.

[31] CERT. Advisory CA-1997-27: FTP Bounce, 1997. `http://www.cert.org/advisories/CA1997-27.html`.

[32] CERT. Advisory CA-2001-02: Multiple Vulnerabilities in BIND, 2001. `http://www.cert.org/advisories/CA-2001-02.html`.

[33] CERT. Advisory CA-2001-19: "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL, 2001. `http://www.cert.org/advisories/CA-2001-19.html`.

[34] Peter Cheeseman and John Stutz. Bayesian Classification (AutoClass): Theory and Results. In Fayyad et al. [64].

[35] David W. Cheung, H.Y. Hwang, Ada W. Fu, and Jiawei Han. Efficient Rule-Based Attribute-Oriented Induction for Data Mining. *Journal of Intelligent Information Systems*, 15(2):175–200, 2000.

[36] William S. Cleveland. *Visualizing Data*. Hobart Press, 1993.

[37] Chris Clifton and Gary Gengo. Developing Custom Intrusion Detection Filters Using Data Mining. In *Military Communications International Symposium (MILCOM2000)*, October 2000.

[38] William W. Cohen. Fast Effective Rule Induction. In *Proceedings 12th International Conference on Machine Learning*, pages 115–123, 1995.

[39] R.M. Cormack. A Review of Classification (with Discussion). *Journal of the Royal Statistical Society A*, 134:321–367, 1971.

[40] Frédéric Cuppens. Managing Alerts in a Multi-Intrusion Detection Environment. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*, pages 22–31, 2001.

[41] Frédéric Cuppens and Alexandre Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA*, 2002.

[42] Marc Dacier and Klaus Julisch. Method, Computer Program Element, and System for Processing Alarms Triggered by a Monitoring System, 2002. Patent, filed as Dockets CH920010041US1 and CH920010041EP1.

[43] Marc Dacier and Klaus Julisch. Method, Computer Program Element, and System for Processing Alarms Triggered by a Monitoring System, 2002. Patent, filed as Dockets CH920010042US1 and CH920010042EP1.

[44] Oliver Dain and Robert K. Cunningham. Fusing Heterogeneous Alert Streams into Scenarios. In Barbará and Jajodia [17].

[45] William H.E. Day. Complexity Theory: An Introduction for Practitioners of Classification. In Arabie et al. [12], pages 199–233.

[46] Hervé Debar, Marc Dacier, Medhi Nassehi, and Andreas Wespi. Fixed versus. Variable-Length Patterns for Detecting Suspicious Process Behavior. In *Proceedings of the 5th European Symposium on Research in Computer Security (ESORICS)*, pages 1–15, 1998.

[47] Hervé Debar, Marc Dacier, and Andreas Wespi. A Revised Taxonomy for Intrusion Detection Systems. *Annales des Télécommunications*, 55(7–8):361–378, 2000.

[48] Hervé Debar and Andreas Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *Proceedings of the 4th Workshop on Recent Advances in Intrusion Detection (RAID)*, LNCS, pages 85–103. Springer Verlag, 2001.

[49] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum Likelihood From Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[50] Edwin Diday. Orders and Overlapping Clusters by Pyramids. In J. De Leeuw, W. Heiser, J. Meulman, and F. Critchley, editors, *Multidimensional Data Analysis*, pages 201–234. DSWO Press, Leiden, The Netherlands, 1986.

[51] Dave Dittrich. Distributed Denial of Service (DDoS) Attacks/Tools, 2002. `http://staff.washington.edu/dittrich/misc/ddos/`.

[52] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Proceedings of the 12th International Conference on Machine Learning*, pages 194–202, 1995.

[53] Norman R. Draper and Harry Smith. *Applied Regression Analysis*. Wiley-Interscience, 3rd edition, 1998.

[54] Richard C. Dubes. Cluster Analysis and Related Issues. In C.H. Chen, L.F. Pau, and P.S.P Wang, editors, *Handbook of Pattern Recognition and Computer Vision*, pages 3–32. World Scientific Publisher, 2nd edition, 1998.

[55] Richard C. Dubes and Anil K. Jain. Clustering Techniques: The User's Dilemma. *Pattern Recognition*, 8:247–260, 1976.

[56] Richard C. Dubes and Anil K. Jain. Validity Studies in Clustering Methodologies. *Pattern Recognition*, 11:235–254, 1979.

[57] Richard C Dubes and Anil K. Jain. Clustering Methodologies in Exploratory Data Analysis. In Marshall C. Yovits, editor, *Advances in Computers*, volume 19, pages 113–228. Academic Press, New York, 1980.

[58] Jennifer G. Dy and Carla E. Brodley. Feature Subset Selection and Order Identification for Unsupervised Learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 247–254, 2000.

[59] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 1994.

[60] Michael Erlinger and Stuart Staniford-Chen. Intrusion Detection Exchange Format (idwg). `http://www.ietf.org/html.charters/idwg-charter.html`.

[61] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[62] Vladimir Estivill-Castro. Why so Many Clustering Algorithms – A Position Paper. *SIGKDD Explorations*, 4(1):65–75, 2002.

[63] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From Data Mining to Knowledge Discovery: An Overview. In Fayyad et al. [64].

[64] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, 1996.

[65] Anuska Ferligoj and Vladimir Batagelj. Some Types of Clustering with Relational Constraints. *Psychometrika*, 48(4):541–552, 1983.

[66] E. Filsinger, J. Faulkner, and R. Warland. Empirical Taxonomy of Religious Individuals: An Investigation Among College Students. *Sociological Analysis*, 40:136–146, 1979.

[67] Douglas H. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2:139–172, 1987.

[68] Douglas H. Fisher. Noise-Tolerant Conceptual Clustering. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 825–830, 1989.

[69] Lloyd Fisher and John W. Van Ness. Admissible Clustering Procedures. *Biometrika*, 58:91–104, 1971.

[70] D. Foti, D. Lipari, Clara Pizzuti, and Domenico Talia. Scalable Parallel Clustering for Data Mining on Multicomputers. In *Proceedings of the 15th International Parallel and Distributed Processing Workshop*, LNCS, pages 390–398, 2000.

[71] Chris Fraley and Adrian E. Raftery. How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis. *The Computer Journal*, 41(8):578–588, 1998.

[72] Fyodor. `http://www.insecure.org/nmap`.

[73] Venkatesh Ganti. CACTUS – Clustering Categorical Data Using Summaries, 1999. `http://www.cs.wisc.edu/~vganti/cactus/cactus-dba.ppt`.

[74] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. CACTUS – Clustering Categorical Data Using Summaries. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery in Databases (SIGKDD)*, pages 73–83, 1999.

[75] Minos Garofalakis and Rajeev Rastogi. Data Mining Meets Network Management: The Nemesis Project. In *Proceedings of the ACM SIGMOD International Workshop on Research Issues in Data Mining and Knowledge Discovery*, May 2001.

[76] Ram Gnanadesikan, Jon R. Kettenring, and James M. Landwehr. Interpreting and Assessing the Results of Cluster Analyses. *Bulletin of the International Statistical Institute*, 47(2):451–463, 1977.

[77] L.A. Goodman and W.H. Kruskal. Measures of Association for Cross-Classifications. *Journal of the American Statistical Association*, 49:732–764, 1954.

[78] Allan D. Gordon. Hierarchical Classification. In Arabie et al. [12], pages 65–121.

[79] Allan D. Gordon. Null Models in Cluster Validation. In W. Gaul and D. Pfeifer, editors, *From Data to Knowledge: Theoretical and Practical Aspects of Classification, Data Analysis, and Knowledge Organization*, pages 32–44. Springer-Verlag, Berlin, 1996.

[80] Allan D. Gordon. Cluster Validation. In C. Hayashi, N. Ohsumi, K. Yajima, Y. Tanaka, H.H. Bock, and Y. Baba, editors, *Data Science, Classification, and Related Methods*, pages 22–39. Springer-Verlag, 1998.

[81] Allan D. Gordon. *Classification*. Chapman and Hall, 1999.

[82] Robert Grossman, Simon Kasif, Reagan Moore, David Rocke, and Jeff Ullman. Data Mining Research: Opportunities and Challenges. Technical

report, Workshop on Managing and Mining Massive and Distributed Data (M3D2), 1998.

[83] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[84] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. *Information Systems*, 25(5):345–366, 2000.

[85] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On Clustering Validation Techniques. *Journal of Intelligent Information Systems (JIIS)*, 17(2-3):107–145, 2001.

[86] Jiawei Han, Yandong Cai, and Nick Cercone. Knowledge Discovery in Databases: An Attribute-Oriented Approach. In *Proceedings of the 18th International Conference on Very Large Databases*, pages 547–559, 1992.

[87] Jiawei Han, Yandong Cai, and Nick Cercone. Data-Driven Discovery of Quantitative Rules in Relational Databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(1):29–40, 1993.

[88] Jiawei Han and Yongjan Fu. Exploration of the Power of Attribute-Oriented Induction in Data Mining. In Fayyad et al. [64].

[89] Jiawei Han and Yongjian Fu. Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 157–168, 1994.

[90] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publisher, 2000.

[91] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, 2001.

[92] Pierre Hansen, Brigitte Jaumard, and Nenad Mladenovic. How to Choose K Entries Among N. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 19:105–116, 1995.

[93] Robin Hanson, John Stutz, and Peter Cheeseman. Bayesian Classification Theory. Technical Report FIA-90-12-7-01, NASA Ames Research Center, 1991.

[94] Joseph L. Hellerstein and Sheng Ma. Mining Event Data for Actionable Patterns. In *The Computer Measurement Group*, 2000.

[95] Alexander Hinneburg and Daniel A. Keim. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In *Proceedings of the 4th ACM International Conference on Knowledge Discovery and Data Mining*, pages 58–65, 1998.

[96] J.S. Urban Hjorth. *Computer Intensive Statistical Methods: Validation, Model Selection and Bootstrap*. CRC Press, 1993.

[97] Frank Hoeppner, Rudolf Kruse, Frank Klawonn, and Thomas Runkler. *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis, and Image Recognition*. John Wiley & Sons, 1999.

[98] K. Houck, Seraphin B. Calo, and Allan J. Finkel. Towards a Practical Alarm Correlation System. In Adarshpal S. Sethi, Yves Raynaud, and Fabienne Faure-Vincent, editors, *Proceedings of the 4th International Symposium on Integrated Network Management*, pages 226–237. Chapman & Hall, 1995.

[99] John D. Howard. *An Analysis of Security Incidents on the Internet*. PhD thesis, Carnegie Mellon University, 1997.

[100] Zhexue Huang. A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining. In *Proceedings of the ACM SIGMOD International Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1997.

[101] Lawrence J. Hubert and Phipps Arabie. Comparing Partitions. *Journal of Classification*, 2:193–218, 1985.

[102] Koral Ilgun, Richard A. Kemmerer, and Phillip A. Porras. State Transition Analysis: A Rule-Based Intrusion Detection System. *IEEE Transactions on Software Engineering*, 21(3):181–199, 1995.

[103] Koral Ilung. USTAT: A Real-Time Intrusion Detection System for UNIX. In *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA*, pages 16–28, 1993.

[104] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[105] Anil K. Jain, Musti N. Murty, and Patrick J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[106] Gabriel Jakobson and Mark D. Weissman. Alarm Correlation. *IEEE Network*, 7(6):52–59, 1993.

[107] Gabriel Jakobson and Mark D. Weissman. Real-Time Telecommunication Network Management: Extending Event Correlation With Temporal Constraints. In Adarshpal S. Sethi, Yves Raynaud, and Fabienne Faure-Vincent, editors, *Proceedings of the 4th International Symposium on Integrated Network Management*, pages 290–301. Chapman & Hall, 1995.

[108] Nicholas Jardine and Robin Sibson. *Mathematical Taxonomy*. John Wiley & Sons, 1971.

[109] Harold S. Javitz and Alfonso Valdes. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA*, pages 316–326. SRI International, 1991.

[110] Klaus Julisch. Dealing with False Positives in Intrusion Detection. In *Extended Abstract at the 3nd Workshop on Recent Advances in Intrusion Detection (RAID)*, 2000. `http://www.raid-symposium.org/raid2000/program.html`.

[111] Klaus Julisch. Mining Alarm Clusters to Improve Alarm Handling Efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*, pages 12–21, December 2001.

[112] Klaus Julisch. Data Mining for Intrusion Detection: A Critical Review. In Barbará and Jajodia [17].

[113] Klaus Julisch. Clustering Intrusion Detection Alarms to Support Root Cause Analysis. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):1–29, 2003.

[114] Klaus Julisch and Marc Dacier. Mining Intrusion Detection Alarms for Actionable Knowledge. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining*, pages 366–375, July 2002.

[115] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar. CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling. *IEEE Computer*, 32(8):68–75, 1999.

[116] Stefan Kätker and Martin Paterok. Fault Isolation and Event Correlation for Integrated Fault Management. In Aurel Lazar, Roberto Saracco, and Rolf Sadler, editors, *Proceedings of the 5th International Symposium on Integrated Network Management*, pages 625–637. Chapman & Hall, 1997.

[117] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[118] David G. Kleinbaum, Lawrence L. Kupper, Keith E. Muller, and Azhar Nizati. *Applied Regression Analysis and Multivariable Methods*. Brooks/Cole Pub Co, 3rd edition, 1997.

[119] Mika Klemettinen. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Data*. PhD thesis, University of Helsinki (Finland), 1999.

[120] Mika Klemettinen, Heikki Mannila, P. Ronkainen, Hannu Toivonen, and A.I. Verkamo. Finding Interesting Rules from Large Sets of Discovered Association Rules. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pages 401–407, 1994.

[121] Ivan V. Krsul. *Software Vulnerability Analysis*. PhD thesis, Purdue University, 1998.

[122] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, 1995.

[123] Kwok-Yan Lam, Lucas Hui, and Siu-Leung Chung. A Data Reduction Method for Intrusion Detection. *Journal of Systems and Software*, 33:101–108, 1996.

[124] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi. A Taxonomy of Computer Program Security Flaws. *ACM Computing Surveys*, 26(3):211–254, 1994.

[125] Terran Lane and Carla E. Brodley. Temporal Sequence Learning and Data Reduction for Anomaly Detection Lane. *ACM Transactions on Information and System Security*, 2(3):295–331, 1999.

[126] Linda Lankewicz and Mark Benard. Real-Time Anomaly Detection Using a Non-Parametric Pattern Recognition Approach. In *Proceedings of the 7th Annual Computer Security Applications Conference*, December 1991.

[127] Jean-Claude Laprie, editor. *Dependability: Basic Concepts and Terminology*, volume 5 of *Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag, Vienna, 1992.

[128] Robert J. Latino and Kenneth Latino. *Root Cause Analysis: Improving Performance for Bottom Line Results*. CRC Press, LLC, 2002.

[129] Wenke Lee and Salvatore J. Stolfo. A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):227–261, 2000.

[130] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. Data Mining Approaches for Intrusion Detection. In *Proceedings of the 7th USENIX Security Symposium (SECURITY '98)*, pages 120–132, 1997.

[131] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A Data Mining Framework for Building Intrusion Detection Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[132] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. Mining in a Data-flow Environment: Experience in Network Intrusion Detection. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)*, pages 114–124, 1999.

[133] Marcus Leech, Matt Ganis, Ying-Da Lee, Ron Kuris, David Koblas, and LaMont Jones. SOCKS Protocol Version 5. Request for Comments (RFC) 1928, 1996.

[134] Lundy M. Lewis. A Case-Based Reasoning Approach to the Resolution of Faults in Communication Networks. In H.-G. Hegering and Y. Yemini, editors, *Proceedings of the 3th International Symposium on Integrated Network Management*, pages 671–682. North Holland, 1993.

[135] Dekang Lin. An Information-Theoretic Definition of Similarity. In *Proceedings of the 15th International Conference on Machine Learning*, pages 296–304, 1998.

[136] Richard Lippmann, Seth Webster, and Douglas Stetson. The Effect of Identifying Vulnerabilities and Patching Software on the Utility of Network Intrusion Detection. In *Proceedings of the 5th Workshop on Recent Advances in Intrusion Detection (RAID)*, LNCS 2516, pages 307–326. Springer Verlag, 2002.

[137] Bing Liu and Wynne Hsu. Post-Analysis of Learned Rules. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 828–834, 1996.

[138] A.D. Livingston, G. Jackson, and K. Priestley. *Root Causes Analysis: Literature Review*. HSE Books, 2001. Available at `http://www.hsebooks.co.uk/`.

[139] Yijun Lu. Concept Hierarchy in Data Mining: Specification, Generation, and Implementation. Master's thesis, Simon Fraser University, Canada, 1997.

[140] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In L.M. Le Cam and J. Neyman, editors, *5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[141] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz. A Data Mining Analysis of RTID Alarms. *Computer Networks*, 34(4), October 2000.

[142] Heikki Mannila and Hannu Toivonen. Discovering Generalized Episodes Using Minimal Occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 146–151, 1996.

[143] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[144] John McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[145] Robert M. McIntyre and Roger K. Blashfield. A Nearest-Centroid Technique for Evaluating the Minimum-Variance Clustering Procedure. *Multivariate Behavioral Research*, 15:225–238, 1980.

[146] Ryszard S. Michalski and Robert E. Stepp. Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(4):396–410, 1983.

[147] Renée J. Miller and Yuping Yang. Association Rules over Interval Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 452–461, 1997.

[148] Glenn W. Milligan. An Examination of the Effect of Six Types of Error Perturbation on Fifteen Clustering Algorithms. *Psychometrika*, 45(3):325–342, 1980.

[149] Glenn W. Milligan. A Monte Carlo Study of Thirty Internal Criterion Measures for Cluster Analysis. *Psychometrika*, 46(2):187–199, 1981.

[150] Glenn W. Milligan. A Review of Monte Carlo Tests of Cluster Analysis. *Multivariate Behavioral Research*, 16:379–407, 1981.

[151] Glenn W. Milligan. Clustering Validation: Results and Implications for Applied Analyses. In Arabie et al. [12], pages 341–375.

[152] Glenn W. Milligan and Martha C. Cooper. An Examination of Procedures for Determining the Number of Clusters in a Data Set. *Psychometrika*, 50(2):159–179, 1985.

[153] Glenn W. Milligan and Martha C. Cooper. Methodology Review: Clustering Methods. *Applied Psychological Measurements*, 11:329–354, 1987.

[154] Glenn W. Milligan, S.C. Soon, and Lisa M. Sokol. The Effect of Cluster Size, Dimensionality, and the Number of Clusters on Recovery of True Cluster Structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(1):40–47, 1983.

[155] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[156] Leslie C. Morey, Roger K. Blashfield, and Harvey A. Skinner. A Comparison of Cluster Analysis Techniques Within a Sequential Validation Framework. *Multivariate Behavioral Research*, 18:309–329, 1983.

[157] Abdelaziz Mounji. *Languages and Tools for Rule-Based Distributed Intrusion Detection*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix Namur (Belgium), 1997.

[158] Ravi Mukkamala, Jason Gagnon, and Sushil Jajodia. Integrating Data Mining Techniques with Intrusion Detection Methods. In *Proceedings of the 13th IFIP WG11.3 Working Conference on Database Security*, pages 33–46, July 1999.

[159] Fionn Murtagh. Neural Networks for Clustering. In Arabie et al. [12], pages 235–269.

[160] Richard E. Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. John Wiley & Sons, 1990.

[161] John W. Van Ness. Admissible Clustering Procedures. *Biometrika*, 60:422–424, 1973.

[162] Raymond T. Ng and Jiawei Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*, pages 144–155, 1994.

[163] Peng Ning, Sushil Jajodia, and Xiaoyang Wang. Abstraction-Based Intrusion Detection in Distributed Environments. *ACM Transactions on Information and System Security (TISSEC)*, 4(4):407–452, 2001.

[164] Yossi A. Nygate. Event Correlation Using Rule and Object Based Techniques. In Adarshpal S. Sethi, Yves Raynaud, and Fabienne Faure-Vincent, editors, *Proceedings of the 4th International Symposium on Integrated Network Management*, pages 278–289. Chapman & Hall, 1995.

[165] David A. Ohsie. *Modeled Abductive Inference for Event Management and Correlation*. PhD thesis, Columbia University, 1998.

[166] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[167] Mark Paradies and David Busch. Root Cause Analysis at Savannah River Plant. In *Proceedings of the IEEE Conference on Human Factors and Power Plants*, pages 479–483, 1988.

[168] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[169] Yun Peng and James A. Reggia. A Probabilistic Causal Model for Diagnostic Problem Solving — Part I: Diagnostic Strategy. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(3):395–404, 1987.

[170] Yun Peng and James A. Reggia. A Probabilistic Causal Model for Diagnostic Problem Solving — Part I: Integrating Symbolic Causal Inference with Numeric Probabilistic Inference. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(2):146–162, 1987.

[171] Gregory Piatetsky-Shapiro. Software for Visualization. `http://www.kdnuggets.com/software/visualization.html`.

[172] Leonard Pitt and Robert E. Reinke. Criteria for Polynomial Time (Conceptual) Clustering. *Machine Learning*, 2(4):371–396, 1987.

[173] Leonid Portnoy, Eleazar Eskin, and Salvatore J. Stolfo. Intrusion Detection with Unlabeled Data Using Clustering. In Barbará and Jajodia [17].

[174] David Powell and Robert Stroud. Architecture and Revised Model of MAFTIA. Technical Report CS-TR-749, University of Newcastle upon Tyne, 2001.

[175] Katherine E. Price. Host-Based Misuse Detection and Conventional Operating Systems' Audit Data Collection. Master's thesis, Purdue University, 1997.

[176] Thomas H. Ptacek and Timothy N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., January 1998.

[177] John R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[178] Roy Rada and Ellen Bicknell. Ranking Documents with a Thesaurus. *Journal of the American Society for Information Science*, 40(5):304–310, 1989.

[179] Roy Rada, Hafedh Mill, Ellen Bicknell, and Maria Blettner. Development and Application of a Metric on Semantic Nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1):17–30, 1989.

[180] Philip Resnik. Semantic Similarity in a Taxonomy: An Information-Based Measure and Its Application to Problems of Ambiguity in Natural Language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999.

[181] Isidore Rigoutsos and Aris Floratos. Combinatorial Pattern Discovery in Biological Sequences: The TEIRESIAS Algorithm. *Bioinformatics*, 14(1):55–67, 1998.

[182] Enrique R. Ruspini. A New Approach to Clustering. *Information and Control*, 15(1):22–32, 1969.

[183] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag. A High-Performance Network Intrusion Detection System. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 8–17, November 1999.

[184] Shokri Z. Selmi and M.A. Ismail. K-Means Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optima. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):81–87, 1984.

[185] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. WaveCluster: A Wavelet-Based Clustering Approach for Spatial Data in Very Large Databases. *International Journal on Very Large Databases*, 8(4):289–304, 2000.

[186] Roger N. Shepard and Phipps Arabie. Additive Clustering: Representation of Similarities as Combinations of Discrete Overlapping Properties. *Psychological Review*, 86:87–123, 1979.

[187] Abraham Silberschatz and Alexander Tuzhilin. On Subjective Measures of Interestingness in Knowledge Discovery. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 275–281, 1995.

[188] Stephen E. Smaha. Haystack: An Intrusion Detection System. In *Proceedings of the 4th IEEE Aerospace Computer Security Applications Conference, Orlando, FL*, pages 37–44, December 1988.

[189] Padhraic Smyth. Breaking out of the Black-Box: Research Challenges in Data Mining. In *Proceedings of the ACM SIGMOD International Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'01)*, 2001.

[190] Ramakrishnan Srikant and Rakesh Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pages 1–12, June 1996.

[191] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical Automated Detection of Stealthy Portscans. In *Proceedings of the ACM Computer and Communications Security IDS Workshop*, pages 1–7, 2000.

[192] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, 2002.

[193] Luis Talavera. Dependency-Based Feature Selection for Clustering Symbolic Data. *Intelligent Data Analysis*, 4(1):19–28, 2000.

[194] Luis Talavera and Javier Béjar. Integrating Declarative Knowledge in Hierarchical Clustering Tasks. In *Proceedings of th 3rd International Symposium on Intelligent Data Analysis*, pages 211–222, 1999.

[195] Luis Talavera and Javier Béjar. Generality-Based Conceptual Clustering with Probabilistic Concepts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):196–206, 2001.

[196] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall International, Inc., 1996.

[197] Henry S. Teng, Kaihu Chen, and Stephen C.Y. Lu. Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequential Patterns. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA*, pages 278–284, May 1990.

[198] H. S. Vaccaro and Gunar E. Liepins. Detection of Anomalous Computer Session Activity. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA*, pages 280–289, May 1989.

[199] Alfonso Valdes and Keith Skinner. Probabilistic Alert Correlation. In *Proceedings of the 4th Workshop on Recent Advances in Intrusion Detection (RAID)*, LNCS, pages 54–68. Springer Verlag, 2001.

[200] Kiri Wagstaff and Claire Cardie. Clustering with Instance-Level Constraints. In *Proceedings of the 17th International Conference on Machine Learning*, pages 1103–1110, 2000.

[201] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting Intrusions Using System Calls: Alternative Data Models. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA*, pages 133–145, May 1999.

[202] Shaula Yemini, Shmuel Kliger, Eyal Mozes, Yechiam Yemini, and David A. Ohsie. High Speed & Robust Event Correlation. *IEEE Communications Magazine*, 34(5):82–90, 1996.

[203] Osmar R. Zaiane, Andrew Foss, Chi-Hoon Lee, and Weinan Wang. On Data Clustering Analysis: Scalability, Constraints, and Validation. In *Proceedings of the 6th Pacific-Asian Conference on Advances in Knowledge Discovery and Data Mining*, pages 28–39, 2002.

[204] Diego Zamboni. *Using Internal Sensors for Computer Intrusion Detection*. PhD thesis, Purdue University, 2001.

[205] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 103–114, 1996.

[206] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A New Data Clustering Algorithm and Its Applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.