Clustering Intrusion Detection Alarms to Support Root Cause Analysis

KLAUS JULISCH
IBM Research, Zurich Research Laboratory

It is a well-known problem that intrusion detection systems overload their human operators by triggering thousands of alarms per day. This paper presents a new approach for handling intrusion detection alarms more efficiently. Central to this approach is the notion that each alarm occurs for a reason, which is referred to as the alarm's *root causes*. This paper observes that a few dozens of rather persistent root causes generally account for over 90% of the alarms that an intrusion detection system triggers. Therefore, we argue that alarms should be handled by identifying and removing the most predominant and persistent root causes. To make this paradigm practicable, we propose a novel alarm-clustering method that supports the human analyst in identifying root causes. We present experiments with real-world intrusion detection alarms to show how alarm clustering helped us identify root causes. Moreover, we show that the alarm load decreases quite substantially if the identified root causes are eliminated so that they can no longer trigger alarms in the future.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g. Firewalls)*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*; D.4.6 [**Operating Systems**]: Security and Protection; H.2.8 [**Database Management**]: Database Applications—*Data mining*; I.2.6 [**Artificial Intelligence**]: Learning—*Concept learning*

General Terms: Algorithms, Management, Security

Additional Key Words and Phrases: Intrusion detection, root cause analysis, data mining, cluster analysis, false positives

1. INTRODUCTION

Over the past ten years, the number as well as the severity of network-based computer attacks have significantly increased [Allen et al. 2000]. As a consequence, classic information security technologies such as authentication and cryptography have gained in importance. Simultaneously, intrusion detection has emerged as a new and potent approach to protect information systems [Bace 2000; Debar et al. 2000]. In this approach, so-called *intrusion*

An earlier version of this paper appeared as "Mining Alarm Clusters to Improve Alarm Handling Efficiency" in *Proceedings of the 17th Annual Computer Security Applications Conference* (*ACSAC 2001*), New Orleans, LA., December. IEEE Computer Society, Los Alamitos, CA, 2001.

Author's address: K. Julisch, IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland; email: kju@zurich.ibm.com.

detection systems (IDSs) are used to monitor information 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 [Broderick 1998; Manganaris et al. 2000] as well as researchers [Axelsson 2000; Bloedorn et al. 2000; Clifton and Gengo 2000; Julisch 2001] 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 [Broderick 1998; Dain and Cunningham 2002; Manganaris et al. 2000]. Tools to automate alarm investigation are being developed [Dain and Cunningham 2002; Debar and Wespi 2001; Valdes and Skinner 2001], but there is currently no silver-bullet solution to this problem.

This paper presents a novel semiautomatic approach for handling intrusion detection alarms 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 occurs. We have made the key observation that in most environments, there is a relatively small number of highly *predominant* root causes. Specifically, we have observed that a few dozens of root causes generally account for over 90% of all alarms. Moreover, these root causes tend to be *persistent*, that is, they do not disappear unless someone removes them. Predominant and persistent root causes are problematic because they trigger an alarm flood that distracts the intrusion detection analyst from spotting real attacks, which tend to be more subtle. We have therefore investigated techniques to efficiently handle such *large* groups of redundant alarms.

The outcome of our research was a semiautomatic process that consists of two steps: step one, which is conventionally called *root cause analysis*, identifies root causes that account for *large* numbers of alarms. Step two removes these root causes and thereby significantly reduces the future alarm load. The experiments presented in this paper show how the one-time effort of identifying and removing root causes pays off by reducing the future alarm load by 87%. This is significant because it enables the intrusion detection analyst to henceforth concentrate on the remaining 13% of alarms. As a general rule, root cause identification and removal should be done roughly once a month in order to keep up with changes in the computing environment.

This paper focuses on the first of the above two steps, that is, on the identification of root causes. To support this step, we have developed a novel alarm-clustering method. The motivation for this method stems from the observation that the alarms of a given root cause are generally "similar" (Section 4.2 will formally define our notion of similarity). Our alarm-clustering method reverses this implication and groups similar alarms together, assuming that these alarms also share the same root cause. For each alarm cluster, a single so-called *generalized alarm* is derived. Intuitively, a generalized alarm is a pattern that an alarm must match in order to belong to the respective cluster. We show that knowledge of generalized alarms vastly simplifies root cause analysis.

An example will clarify the relationship between root causes, alarm clustering, and root cause analysis. Let us consider the root cause of a broken TCP/IP stack, which fragments all outgoing IP traffic and thereby triggers "fragmented IP" alarms. Moreover, let us assume that the broken TCP/IP stack belongs to a popular Web server that is primarily used on workdays. Clearly, all "fragmented IP" alarms have the same source IP address (namely, the IP address of the Web server) and the same source port (namely 80). The targets of the alarms are nonprivileged ports of various Web clients. Given that the Web server is mostly used on workdays, it follows that the majority 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 the 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.

Our alarm-clustering method groups the "fragmented IP" alarms together and reports them by a single generalized alarm. This generalized alarm states that "source port 80 of the Web server triggers many 'fragmented IP' alarms on workdays against nonprivileged ports of Web clients," Clearly, a generalized alarm like this facilitates the identification of root causes, but human expertise is still needed. Therefore, alarm clustering only supports root cause analysis, but does not completely automate it. Moreover, recall that root cause analysis is only the first in a two-step process. The second step is to act upon the identified root causes. Ideally, one would always remove root causes (e.g., by fixing the broken TCP/IP stack). Unfortunately, some root causes are not under our control or they are expensive to remove. Then, custom-made filtering rules (which automatically discard alarms) or correlation rules (which intelligently group and summarize alarms) can be an alternative. Clearly, this second step requires care and human judgment, as well.

The novel contribution of this paper is fourfold: first, we show that a few root causes generally account for the majority of alarms in an alarm log; second, we present a novel alarm-clustering method that groups similar alarms together, and summarizes them by a single generalized alarm; third, we show that knowledge of generalized alarms vastly simplifies the identification of root causes; finally, we show that removing root causes can significantly reduce the future alarm load, thus enabling a more thorough analysis of the remaining alarms.

Paper Overview. The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 defines our terminology. Section 4 formalizes the problem that alarm clustering should solve. Section 5 proposes an algorithm to solve this problem. Section 6 presents our experience with alarm clustering and root cause analysis. Section 7 contains concluding remarks.

2. RELATED WORK

Related work toward "better" IDSs, which trigger less false positives, is described in Section 2.1. Section 2.2 discusses real-time correlation systems for intrusion detection alarms. Section 2.3 surveys how data mining has been used to support the investigation of alarms. Section 2.4 reviews alarm correlation and root cause analysis in the context of network fault management.

2.1 Toward Better IDSs

The intuitively most appealing way of dealing with false positives is to build "better" IDSs, which trigger less false positives. This is a challenging endeavor because false positives are the result of multiple problems, including a lack of suitable audit sources [Price 1997; Ptacek and Newsham 1998], harsh real-time requirements (which preclude a thorough analysis of the audit data) [Ilung 1993; Ptacek and Newsham 1998], the problem that for some events (e.g., failed logins) it is undecidable whether they constitute attacks [Bellovin 1993; Paxson 1999], and the inherent difficulty of writing correct intrusion detection signatures [Kumar 1995; Lee and Stolfo 2000; Mounji 1997; Ning et al. 2001]. A "better" IDS would have to address all these issues, and a small number of research projects have attempted to so.

Examples of IDSs that are less prone to false positives include the embedded detectors technology by Zamboni [2001], a lightweight tool for detecting Web server attacks by Almgren et al. [2000], and a network-based IDS that focuses exclusively on low-level network attacks [Sekar et al. 1999]. Interestingly, all three IDSs share two commonalities: first, they have public signatures that can be tuned to a given environment, and second they are special purpose. *Special purpose IDSs* are tailored toward detecting one class of attacks (e.g., Web server attacks), and they monitor audit sources that are particularly suitable for this task. A drawback of special-purpose IDSs is that they must be combined with other complementary IDSs to obtain comprehensive coverage.

2.2 Alarm Correlation

Alarm correlation systems (ACSs) [Cuppens 2001; Cuppens and Miège 2002; Dain and Cunningham 2002; Debar and Wespi 2001; Staniford et al. 2000; Valdes and Skinner 2001] try to group alarms so that the alarms of the same group pertain to the same phenomenon (e.g., the same attack). In that way, they offer a more condensed view on the security issues raised by an IDS. In particular, they make it easier to distinguish false positives from real security threats. As ACSs operate in real time, they can be viewed as real-time clustering systems (after all, they also group/cluster alarms). However, the clustering method proposed in this paper contrasts along three dimensions with prior work:

Depth of Analysis. Being conceived for off-line usage, our clustering algorithm performs a more thorough analysis than ACSs. For example, consider a phenomenon that only occurs on Saturdays (e.g., false positives due to weekly system backups). Our clustering algorithm 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.

Ease of Use. Some ACSs have dozens of configuration parameters, which take experience to set [Debar and Wespi 2001; Valdes and Skinner 2001]. Other ACSs face a knowledge engineering bottleneck because they require the user to specify correlation rules [Cuppens 2001; Cuppens and Miège 2002]. The ACS by Dain and Cunningham [2002] learns correlation rules from the user. To this end, the user has to manually correlate alarms, so the system can learn his or her ability. Unfortunately, manual alarm correlation is difficult and time consuming. As will become apparent throughout this paper, our clustering algorithm is easy and intuitive to use.

Bias. ACSs are generally optimized to build alarm groups that correspond to *attacks*. For example, some ACSs implement special techniques to deal with spoofed source addresses and with alarms that belong to larger multistage attacks [Cuppens and Miège 2002; Dain and Cunningham 2002; Valdes and Skinner 2001]. Other ACSs reassess the severity of alarm groups and discard alarm groups that are considered benign [Debar and Wespi 2001; Staniford et al. 2000]. Virtually all publications use only attacks to test the proposed ACSs. By contrast, our clustering method is designed to find large groups of mostly false positives. It is questionable if today's ACSs with their attack-centric bias are suitable for finding such groups.

2.3 Data Mining

The idea of using data mining to support alarm investigation is not new. Manganaris et al. [2000] mine association rules over alarm bursts. Subsequently, alarms that are consistent with these association rules are deemed "normal" and get discarded. The risk of discarding true positives is not considered in this work, whereas bounding this risk is central to our approach. Clifton and Gengo [2000] use episode mining to guide the construction of custom-made filtering rules. The present paper pursues a similar idea, but mines alarm clusters rather than episode rules. Finally, the previously mentioned work by Dain and Cunningham [2002] is also relevant in this context as it uses data mining techniques to learn correlation rules from hand-labeled training examples.

Some researchers have used data mining to build IDSs more systematically. For example, Barbará et al. [2001] use incremental data mining techniques to detect anomalous network traffic patterns in real time. Lee and Stolfo [2000] use data mining for feature construction and training of classifiers that detect intrusions. A recent book edited by Barbará and Jajodia [2002] surveys these projects, and offers a general treatment of data mining in computer security. Data mining for fraud detection is investigated by Fawcett and Provost [1997] and by Chan and Stolfo [1998]. Finally, in the world of telecommunication networks, Klemettinen [1999] uses association rules and episode rules to support the development of alarm correlation systems. Hellerstein and Ma [2000] pursue the same goal by means of visualization, periodicity analysis, and m-patterns (a variant of association rules requiring mutual implication).

2.4 Network Fault Management

In network fault management [Bouloutas et al. 1994; Houck et al. 1995; Jakobson and Weissman 1993, 1995; Lewis 1993; Nygate 1995; Ohsie 1998; Yemini et al. 1996], 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 fault management is to evaluate these alarms and to pinpoint the original problems (the so-called root causes).

Identifying root causes is an instance of the general problem of abductive inference [Ohsie 1998; Peng and Reggia 1987a, 1987b]. *Abductive inference* is the process of reasoning from effects (i.e., alarms) 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 [Bouloutas et al. 1994; Houck et al. 1995]. Other systems require the user to encode his or her knowledge about root cause analysis in expert system rules [Jakobson and Weissman 1993, 1995; Nygate 1995]. 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 [Lewis 1993] or codebooks [Yemini et al. 1996].

All of the above research is not directly applicable to intrusion detection because the notions of dependence and cause–effect propagation are not easily transferable. Moreover, network management systems can only diagnose *known* root causes. By contrast, this paper describes a clustering method that supports the discovery of new, previously unknown root causes.

3. DEFINITIONS AND NOTATIONS

This section defines concepts that are central to this paper, including the notions of root causes, root cause analysis, alarms, alarm logs, generalized alarms, and generalization hierarchies.

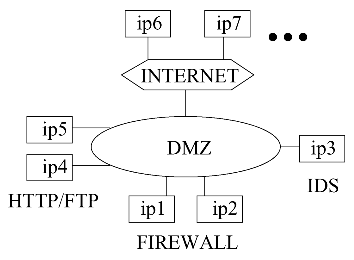
Root Causes and Root Cause Analysis. The *root cause* of an alarm, as defined in the Introduction, is the reason for which it is triggered. Another useful mental construct is that root causes are problems that affect components and cause them to trigger alarms. For example, a failure can affect the implementation of a TCP/IP stack, and cause it to fragment all outbound IP traffic, which triggers "fragmented IP" alarms. Similarly, a worm is a root cause that affects a set of hosts and causes them to trigger alarms when the worm spreads. *Root cause analysis* is the task of identifying root causes as well as the components they affect. We do not attempt to formally define root causes or root cause analysis, because such an attempt seems fruitless. In fact, in the dependability field, the term *fault* denotes a concept that is very similar to a root cause, and the dependability notion of *fault diagnosis* corresponds to root cause analysis [Laprie 1992; Powell and Stroud 2001]. Neither faults nor fault diagnoses have been formally defined. Therefore, we consider it unlikely that the intrusion detection equivalents of these terms possess formal definitions.

Alarms, Alarm Logs, and Generalized Alarms. Intrusion detection systems trigger *alarms* to report presumed security violations. This paper models alarms as tuples over the Cartesian product $\text{dom}(A_1) \times \cdots \times \text{dom}(A_n)$, 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. The value that attribute A_i assumes in alarm $\mathbf{a}$ is denoted by $\mathbf{a}[A_i]$. This paper models *alarm logs* as sets of alarms. This model is correct because alarms are implicitly ordered by virtue of their time-stamps; moreover, unique alarm identifiers can be used to guarantee that all alarms are pairwise distinct.

A *generalized attribute value* is a concept name that represents a subset of an attribute domain $\text{dom}(A_i)$, $i \in \{1, \ldots, n\}$. For example, the generalized attribute value *Web-server* might represent the subset of IP addresses that host Web servers. Similarly, the generalized attribute value *Weekdays* can be defined to comprise the subset of time-stamps that fall on weekdays. The *extended domain* $\text{Dom}(A_i)$ of an 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. Finally, a *generalized alarm* is a tuple in $[\text{Dom}(A_1) \times \cdots \times \text{Dom}(A_n)] \smallsetminus [\text{dom}(A_1) \times \cdots \times \text{dom}(A_n)]$. Note that a generalized alarm $\mathbf{g}$ *models* the set $\{\mathbf{a} \mid \forall A_i : (\mathbf{a}[A_i] = \mathbf{g}[A_i] \vee \mathbf{a}[A_i] \in \mathbf{g}[A_i])\}$ of ordinary (i.e., ungeneralized) alarms $\mathbf{a}$.

Generalization Hierarchies. For each attribute A_i, let $\mathcal{G}_i$ be a single-rooted and connected 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* (a.k.a. *is-a* hierarchy or taxonomy). For example, Figure 1(c) shows sample generalization hierarchies for IP addresses and port numbers. 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 a directed path from $\hat{x}$ to x (in symbols: $x \trianglelefteq \hat{x}$). To extend these definitions from attributes to alarms, let $\mathbf{a}, \hat{\mathbf{a}} \in \times_{1 \le i \le n} \text{Dom}(A_i)$ denote two (possibly generalized) alarms. The alarm $\hat{\mathbf{a}}$ is called a *parent* of alarm $\mathbf{a}$ if and only if $\mathbf{a}[A_i] \trianglelefteq \hat{\mathbf{a}}[A_i]$ holds for all attributes A_i. This is denoted by $\mathbf{a} \trianglelefteq \hat{\mathbf{a}}$, and the alarm $\mathbf{a}$ is said to be more *specific* than $\hat{\mathbf{a}}$, while $\hat{\mathbf{a}}$ is called more *abstract* or *general* than $\mathbf{a}$.

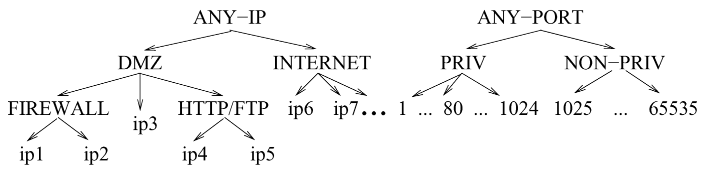
Example. By way of illustration, Figure 1 shows a network topology, a sample alarm log, and sample generalization hierarchies for IP addresses and port



(a) Network

Dest−IP	Dest−Port
ip1	80
ip2	1234
ip2	23
ip4	8080
ip4	80
ip4	21
ip5	80

(b) Sample alarm log



(c) Generalization hierarchies for IP addresses and port numbers

Fig. 1. Network, alarm log, and generalization hierarchies of the running example.

numbers. Let us verify that the following holds:

—Let **a** be the first alarm in the alarm log of Figure 1(b), that is, **a** := (*ip1*, 80). Then, **a**[Dest-IP] = *ip1* 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*, *DMZ*, *INTERNET*, *ANY-IP*} of generalized attribute values). Similarly, the extended domain of port numbers is {1, ..., 65535, *PRIV*, *NON-PRIV*, *ANY-PORT*}.

—According to Figure 1(c), the IP address *ip1* is a firewall, which is a machine in the DMZ, which, more generally, is any IP address. More succinctly, this relationship can be expressed as *ip1* $\trianglelefteq$ *FIREWALL* $\trianglelefteq$ *DMZ* $\trianglelefteq$ *ANY-IP*. Moreover, note that *ip1* $\trianglelefteq$ *ip1* and (*ip1*, 80) $\trianglelefteq$ (*FIREWALL*, *PRIV*) hold.

4. ALARM-CLUSTERING PROBLEMS

In this paper, we use alarm clustering to extract alarm groups, called clusters, that a skilled user can interpret in terms of root causes. Because alarm clusters can be huge, we additionally summarize them by means of generalized alarms, which facilitates their interpretation. It is intuitively clear that the way in which we group alarms has a strong influence on how useful the resulting alarm clusters are. Ideally, alarm clustering should solve the following problem:

Definition 1 (*Exact Alarm-Clustering Problem*). Given an alarm log, the *exact alarm-clustering problem* is to group alarms into *clusters* such that all alarms of a given cluster share the same root cause.

An algorithm that solves the exact alarm-clustering problem eliminates redundancy by grouping alarms that share the same root cause. Furthermore, if one understands the root cause of any alarm in an alarm cluster, then one understands the root causes of all alarms in the cluster.

Unfortunately, there is no algorithmic solution to the exact alarm-clustering problem. In fact, root causes are a model world concept, which only exists in the world of our thinking. Computer programs are not aware of root causes and can therefore not enforce the requirement that all alarms of an alarm cluster must share the same root cause. To illustrate this, let us consider a machine whose broken TCP/IP stack fragments most of the 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 the said machine. Unless substantial background 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 generally, if only an alarm log is given, it is not possible to decide whether two or more alarms have the same root cause. Hence, it is impossible to solve the exact alarm-clustering problem.

Given this negative result, we will next define a variant of the exact alarm clustering problem, which is soluble and yet useful in practice. Section 4.1 uses examples to derive this variant, and Section 4.2 refines it.

4.1 Approximate Alarm-Clustering Problem

This section uses examples to show that many root causes manifest themselves in alarm clusters that have certain structural properties. We are interested in these structural properties because they will lead to the *approximate alarm-clustering problem*, which is to find alarm clusters that have these structural properties. Following the logic of abductive inference (cf. Section 2.4), we then argue that alarm clusters with these structural properties are most likely to be the result of root causes. Abductive arguments like this are very common. 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 clusters having the structural property P, then detection of an alarm clusters having property P is likely to be the result of a root cause. Finally, to determine said structural properties, we consider the following sample root causes:

(1) A HTTP server with a broken TCP/IP stack that fragments outgoing traffic. "Fragmented IP" alarms ensue when the server responds to clients requests.
(2) At one site, a misconfigured secondary DNS server performed half-hourly DNS zone transfers from the 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

Table I. The Generalized Alarms Induced by Nine Sample Root Causes

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

users *simultaneously* request external services, then the firewall will proxy 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) Many network management tools query sensitive MIB variables and thereby trigger IDS alarms. (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) A distributed denial-of-service (DDoS) attack being launched from an external network against a Web hosting site triggered “SYN flooding” alarms.

(9) External Code Red infected machines [CERT 2001] 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 nonprivileged ports. In addition, these alarms always have “fragmented IP” as alarm type. Therefore, the first root cause manifests itself in an alarm cluster that can be modeled by the generalized alarm shown in the first row of Table I. Similarly, the second row of the table shows that the second root cause can be modeled by the generalized alarm of “‘DNS zone transfer’ alarms being triggered from a nonprivileged port of the secondary DNS server against port 53 of the primary DNS server”. Analogously, the remaining rows of Table I show the generalized alarms 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 “Nonpriv.” denotes the set $\{1025, \ldots, 65535\}$ of nonprivileged ports, and the entry "Privileged" stands for the set of privileged ports below 1025.

Another important observation is that most of the above root causes 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 clusters. 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 will abound. Similarly, the misconfigured secondary DNS server triggers one "DNS zone transfer" alarm every 30 mins. This makes approximately 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.

Based on the above examples, we postulate that many root causes manifest themselves in large alarm clusters 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 clusters. In other words, little information is lost when modeling alarm clusters by generalized alarms. To summarize:

PROPOSITION 1 (Alarm Cluster Hypothesis). *Root causes frequently manifest themselves in large alarm clusters that are adequately modeled by generalized alarms.*

The alarm cluster hypothesis is something like a "natural law" of intrusion detection, and can probably not be proven. However, the alarm cluster hypothesis is not completely unexpected, either. In fact, recall from Section 1 that most intrusion detection alarms are false positives. In other words, most alarms are triggered by benign root causes. Now, note that anomaly detection systems only work because benign (or normal) behavior is *systematic* and *repetitive*. In fact, if benign/normal behavior was not systematic, then a model could not capture it; and if it was not repetitive, then models would out-date too fast. Analogously, benign root causes tend to be systematic (which makes generalized alarms an adequate model for their alarms) and repetitive (which causes them to trigger many alarms). Given that benign root causes account for the majority of alarms, we now see that the alarm cluster hypothesis is plausible. Using the abductive argument given at the beginning of this section, we can finally define:

Definition 2 (*Approximate Alarm-Clustering Problem*). Given an alarm log, the *approximate alarm-clustering problem* is to find *large* alarm clusters that are *adequately* modeled by generalized alarms.

4.2 A Framework for Alarm Clustering

A problem with Definition 2 is that it leaves the terms "large" and "adequately" undefined. This section addresses this problem, and formalizes these terms. Before reading on, please recall the definitions of Section 3. In particular, recall that alarms are tuples over the n-dimensional attribute space $\text{dom}(A_1) \times \cdots \times \text{dom}(A_n)$, that $\mathbf{a}[A_i]$ denotes the A_i-value of alarm $\mathbf{a}$, that generalization hierarchies $\mathcal{G}_i$ are single-rooted directed acyclic graphs over the extended attribute domains $\mathrm{Dom}(A_i) := \mathrm{dom}(A_i) \cup \{\text{generalized attribute values for } A_i\}$, and that the relation $\trianglelefteq$ orders attribute values and alarms by their generality.

To formalize the notion of adequacy, we define the *dissimilarity* $d(\cdot,\cdot)$, which takes two alarms as input and returns a numerical measure of how adequately these alarms can be modeled by a single generalized alarm. Dissimilarity is inversely related to similarity, that is, the alarms $\mathbf{a_1}$ and $\mathbf{a_2}$ are similar (and can be adequately modeled by a generalized alarm) if $d(\mathbf{a_1}, \mathbf{a_2})$ is small. To define dissimilarity, we need user-defined generalization hierarchies $\mathcal{G}_i$ for all attributes A_i, $i = 1, \ldots, n$. Section 6.1 considers the problem of defining such generalization hierarchies. For the time being, we simply assume that suitable generalization hierarchies have been defined. For example, one could use the generalization hierarchies of Figure 1(c) for IP addresses and port numbers.

We begin by defining dissimilarity for individual attributes and then generalize it to alarms. To this end, let A_i be an attribute, and let $\mathcal{G}_i$ be the aforementioned generalization hierarchy for A_i, $i = 1, \ldots, n$. The dissimilarity $d(x_1, x_2)$ between any two elements $x_1, x_2 \in \mathrm{Dom}(A_i)$ is the length of the shortest path in $\mathcal{G}_i$ that connects x_1 and x_2 via a common parent p, that is, $d(x_1, x_2) := \min\{\,\delta(x_1, p) + \delta(x_2, p) \mid p \in \mathcal{G}_i,\ x_1 \trianglelefteq p,\ x_2 \trianglelefteq p\,\}$, where $\delta(\cdot,\cdot)$ measures the length of the shortest path between two nodes in $\mathcal{G}_i$. For example, in Figure 1(c), we have $d(\mathit{ip1}, \mathit{ip1}) = 0$, $d(\mathit{ip1}, \mathit{ip4}) = 4$, and $d(\mathit{PRIV}, \mathit{NON\text{-}PRIV}) = 2$. The dissimilarity measure $d(\cdot,\cdot)$ has its origin in the field of information retrieval, where similar measures have been studied [Lin 1998; Rada and Bicknell 1989; Rada et al. 1989; Resnik 1999].

Next, we extend our dissimilarity measure from attributes to alarms. To this end, let $\mathbf{a_1}, \mathbf{a_2} \in \times_{1 \le i \le n} \mathrm{Dom}(A_i)$ denote two (possibly generalized) alarms. The dissimilarity $d(\mathbf{a_1}, \mathbf{a_2})$ between any two alarms $\mathbf{a_1}$ and $\mathbf{a_2}$ is defined as the sum of attribute dissimilarities, that is, $d(\mathbf{a_1}, \mathbf{a_2}) := \sum_{i=1}^{n} d(\mathbf{a_1}[A_i], \mathbf{a_2}[A_i])$. An obvious generalization that we do not pursue further in this paper is to use the weighted sum of attribute dissimilarities as interalarm dissimilarity. By way of illustration, in the sample log of Figure 1(b), we have $d(\,(\mathit{ip1}, 80), (\mathit{ip2}, 1234)\,) = d(\mathit{ip1}, \mathit{ip2}) + d(80, 1234) = 2 + 4 = 6$.

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, let $\mathbf{g} \in \times_{1 \le i \le n} \mathrm{Dom}(A_i)$ be a generalized alarm to which both alarms can be generalized, that is, $\mathbf{a_1}, \mathbf{a_2} \trianglelefteq \mathbf{g}$. Note that $d_i := d(\mathbf{g}, \mathbf{a_i})$, $i = 1, 2$, is the number of times 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 adequately $\mathbf{g}$ models 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 dissimilarity $d(\mathbf{a_1}, \mathbf{a_2})$ 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. Hence, a small dissimilarity value implies that an adequate model exists.

In generalization of the dissimilarity $d(\cdot,\cdot)$, we now define the *heterogeneity* $H(\mathcal{C})$ of an alarm cluster $\mathcal{C}$. Heterogeneity is a function that returns a small value when the cluster $\mathcal{C}$ can be adequately modeled by a generalized alarm. To formally define heterogeneity, let $\mathbf{g}$ be a generalized alarm that is a parent of all alarms in $\mathcal{C}$, that is, $\forall \mathbf{a} \in \mathcal{C} : \mathbf{a} \trianglelefteq \mathbf{g}$. The *average dissimilarity* $\bar{d}(\mathbf{g}, \mathcal{C})$ between $\mathbf{g}$ and $\mathcal{C}$ and the *heterogeneity* $H(\mathcal{C})$ are defined as follows:

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

$$H(\mathcal{C}) := \min \left\{ \bar{d}(\mathbf{g}, \mathcal{C}) \mid \mathbf{g} \in \times_{i=1}^{n} \mathrm{Dom}(A_i),\ \forall \mathbf{a} \in \mathcal{C} : \mathbf{a} \trianglelefteq \mathbf{g} \right\}. \tag{2}$$

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

Definition 3 (*Alarm-Clustering Problem*). Let $\mathcal{L}$ 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* ($\mathcal{L}$, *min_size*, $\mathcal{G}_1, \ldots, \mathcal{G}_n$) is to find a set $\mathcal{C} \subseteq \mathcal{L}$ that *minimizes* the heterogeneity $H(\mathcal{C})$, subject to the constraint that $|\mathcal{C}| \geq$ *min_size* holds. We call $\mathcal{C}$ an *alarm cluster* or *cluster* for short.

In other words, among all sets $\mathcal{C} \subseteq \mathcal{L}$ that satisfy $|\mathcal{C}| \geq$ *min_size*, a set with minimum heterogeneity has to be found. If there are several such sets, then any one of them can be picked. By minimizing heterogeneity, we maximize the adequacy of the generalized alarms that model $\mathcal{C}$. Note that the *min_size* parameter formalizes our hitherto intuitive notion of "largeness," which first appeared in Proposition 4.1. Finally, once the cluster $\mathcal{C}$ has been found, the remaining alarms in $\mathcal{L} \smallsetminus \mathcal{C}$ can be searched for additional clusters.

Some concluding remarks are in order. First, note that we initially defined (alarm) clusters to be groups of alarms that share the same root cause (cf. Definition 1). Henceforth, an (alarm) cluster is a set of alarms as defined in Definition 3. Second, note that stealthy attacks that trigger fewer than *min_size* alarms do not yield any clusters. More generally, solving the alarm-clustering problem will not enable us to discover all kinds of root causes. However, based on its derivation, we know that solving the alarm-clustering problem will enable us to discover a large and practically relevant class of root causes (cf. Proposition 4.1). Finally, it is worth pointing out that alarm clusters have an intensional description in the form of covers. Therefore, covers can be used to summarize and report alarm clusters. Because alarm clusters tend to be huge, it is frequently more practicable to report alarm clusters by means of their covers, rather than by listing their constituent alarms.

5. PRACTICAL ALARM CLUSTERING

This section describes a heuristic algorithm for solving the alarm-clustering problem. In addition, it addresses the issue of setting the *min_size* parameter.

First, however, the following result is of importance:

Proposition 1. *Let $\mathcal{L}$ be an alarm log, min_size $\in \mathbb{N}$ an integer, and $\mathcal{G}_i$, $i = 1, \ldots, n$, a family of generalization hierarchies. The alarm-clustering problem $(\mathcal{L}, min\_size, \mathcal{G}_1, \ldots, \mathcal{G}_n)$ is NP-complete.*

Proof. The proof is obtained by reducing the CLIQUE problem [Papadimitriou 1994] to the alarm-clustering problem. In the CLIQUE problem, we are given a graph G and an integer k. The goal is to decide whether G contains a *k-clique*, that is, a fully connected subgraph of size k. To reduce the CLIQUE problem to the alarm-clustering problem, we assign a separate attribute to each node in G. For each edge in G, we create an alarm. The two attributes that correspond to the end points of the edge are set to 1, while all the other attributes are set to zero. The set of all alarms constitutes the alarm log, *min_size* is set to $\binom{k}{2}$, and the generalization hierarchies define 1 to be the single parent of 0, that is, $\mathcal{G}_i := 1 \longrightarrow 0$ for $i = 1, \ldots, n$.

Let $\mathcal{C}$ be a solution of the alarm-clustering problem that has just been defined. Set $\delta_i := \max\{\mathbf{a}[A_i] \mid \mathbf{a} \in \mathcal{C}\}$ (note that $\delta_i \in \{0, 1\}$ holds). Then, the graph G contains a k-clique if and only if $\sum_{i=1}^{n} \delta_i = k$. Details are given elsewhere [Julisch 2001]. □

Given the need for a scalable solution, Section 5.1 describes a heuristic algorithm for solving the alarm-clustering problem. Section 5.2 discusses extensions to the heuristic algorithm, and Section 5.3 proposes algorithmic support for setting the *min_size* parameter.

5.1 A Heuristic Alarm-Clustering Method

Given the NP completeness of the alarm-clustering problem, we have developed a heuristic algorithm. This algorithm finds clusters $\mathcal{C} \subseteq \mathcal{L}$ that satisfy $|\mathcal{C}| \geq min\_size$, but do not necessarily minimize the heterogeneity $H(\mathcal{C})$. The heuristic algorithm is a variant of attribute-oriented induction (AOI) [Han et al. 1992, 1993], a well-established technique in the field of data mining. Our modifications over the classical AOI are twofold: first, we generalize attributes more conservatively than classical AOI; second, we use a different termination criterion, which is reminiscent of density-based clustering [Agrawal et al. 1998; Han and Kamber 2000].

For the sake of simplicity, we will assume that all generalization hierarchies are trees (we will revisit this assumption in Section 5.2). It follows from this assumption that each alarm cluster possesses a unique cover. Our heuristic algorithm constructs this cover directly, that is, it does *not* make the detour over first finding an alarm cluster and then deriving its cover. Rather, alarm clusters must be determined after the fact by determining for each cover the set of alarms that match it. In more detail, the algorithm starts with the alarm log $\mathcal{L}$ and repeatedly generalizes the alarms in $\mathcal{L}$. Generalizing alarms is done by choosing an attribute A_i and replacing the A_i values of all alarms in $\mathcal{L}$ by their parents in $\mathcal{G}_i$. This process continues until an alarm has been found to which at least *min_size* of the original alarms can be generalized. This alarm constitutes the output of the algorithm.

```
Input:     An alarm clustering problem $(\mathcal{L}, min\_size, \mathcal{G}_1, \ldots, \mathcal{G}_n)$
Output:    A heuristic solution for $(\mathcal{L}, min\_size, \mathcal{G}_1, \ldots, \mathcal{G}_n)$
Algorithm:
  1:  $T := \mathcal{L}$;                                    // Store log $\mathcal{L}$ in table $T$.
  2:  for all alarms $\mathbf{a}$ in $T$ do $\mathbf{a}[count] := 1$;        // Initialize counts.
  3:  while $\forall \mathbf{a} \in T : \mathbf{a}[count] < min\_size$ do {
  4:      Use heuristics to select an attribute $A_i$, $i \in \{1, \ldots, n\}$;
  5:      for all alarms $\mathbf{a}$ in $T$ do              // Generalize attribute $A_i$
  6:          $\mathbf{a}[A_i]$ := father of $\mathbf{a}[A_i]$ in $\mathcal{G}_i$;
  7:      while identical alarms $\mathbf{a}, \mathbf{a}'$ exist do      // Merge identical alarms.
  8:          Set $\mathbf{a}[count] := \mathbf{a}[count] + \mathbf{a}'[count]$ and delete $\mathbf{a}'$ from $T$;
  9:  }
 10:  Output all generalized alarms $\mathbf{a} \in T$ with $\mathbf{a}[count] \geq min\_size$;
```

Fig. 2. Heuristic alarm-clustering algorithm.

Figure 2 shows the pseudocode of the alarm-clustering method. Line 1 copies the alarm log $\mathcal{L}$ into a relational database table T. This table has one attribute (or column) for each alarm attributes A_i. In addition, the table T possesses the integer-valued attribute *count*, which is used for bookkeeping, only. Line 2 sets the count attributes of all alarms to 1. In the lines 3 to 9, the algorithm loops until an alarm **a** has been found, whose count value is at least *min_size*. Line 4 selects an alarm attribute A_i according to a heuristic, which we will describe in a moment. The lines 5 and 6 replace the A_i values of all alarms in T by their parent values in $\mathcal{G}_i$. By doing so, previously distinct alarms can become identical. Two alarms **a** and **a**′ are *identical* 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 7 and 8 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. Moreover, each generalized alarm **a** represents an alarm cluster of size $\mathbf{a}[count]$. To conclude the discussion, we now specify the heuristic that we use in line 4:

Definition 4 (*Attribute Selection Heuristic*). For each attribute A_i, let $F_i := \max\{f_i(v) \mid v \in Dom(A_i)\}$ be the maximum of the function

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

which sums the counts of all alarms $\mathbf{a} \in T$ with $\mathbf{a}[A_i] = v$. Line 4 of Figure 2 selects any attribute A_i whose F_i value is minimal, that is, the F_i value must satisfy $\forall j : F_i \leq F_j$.

The intuition behind this heuristic is that if there is an alarm **a** that satisfies $\mathbf{a}[count] \geq min\_size$, then $F_i \geq f_i(\mathbf{a}[A_i]) \geq min\_size$ holds for all alarm attributes A_i, $i = 1, \ldots, n$. In other words, an alarm **a** with $\mathbf{a}[count] \geq min\_size$ cannot exist (and the algorithm cannot terminate) unless $F_i \geq min\_size$ holds for all attributes A_i. We therefore use it as a heuristic to increase any of the smallest F_i values by generalizing its corresponding attribute A_i. Other heuristics are clearly possible, but the one above performed favorable in informal experiments. Moreover, rather than merely varying the heuristic, one could even conceive a completely different clustering algorithm, for example, one that is based on partitioning or hierarchical clustering [Han and Kamber 2000; Jain and Dubes 1988]. It is a well-known problem of clustering methods that there are few guidelines for choosing the "right" method, let alone for proving the "correctness" of clustering results [Anderberg 1973; Jain and Dubes 1988; Milligan 1996]. We therefore make no formal claims about the above clustering method except that it works well in practice (cf. Section 6).

5.2 DAG-Structured Generalization Hierarchies

This section extends the heuristic alarm-clustering algorithm to support DAG-structured generalization hierarchies. 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 *ip* that simultaneously runs an HTTP server and an FTP server. Accordingly, *HTTP-server* or *FTP-server* are two possible generalizations of *ip*. The following rule assumes that *ip* is the destination IP of alarm **a**, and generalizes *ip* according to the value of the destination port:

```
if a[Destination-port] = 80
   then generalize ip to HTTP-server
   else generalize ip to FTP-server;
```

A similar rule is conceivable for the case that *ip* 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 strategies have been studied in the context of classic AOI [Cheung et al. 2000; Han et al. 1992], and we can directly reuse the solutions proposed there. In the choose-one strategy, the problem of *induction anomaly* needs to be solved [Cheung et al. 2000]. 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. The solution to the induction anomaly problem is to determine the generalization path for all attribute values *before* the first attribute is actually generalized [Cheung et al. 2000]. The explore-all strategy is conceptually easy to implement, as well [Han et al. 1992]. First, line 6 of Figure 2 is replaced by:

```
6.1:   $T := T \setminus \{\mathbf{a}\}$;
6.2:   for all parents $p$ that $\mathbf{a}[A_i]$ has in $\mathcal{G}_i$ do
6.3:   {   $\mathbf{a}' := \mathbf{a}$;   $\mathbf{a}'[A_i] := p$;   $T := T \cup \{\mathbf{a}'\}$;   }
```

In other words, the attribute A_i of alarm **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 8 of Figure 2. 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.

5.3 Algorithmic Support for Setting the *min_size* Parameter

Currently, the user is the only authority over the *min_size* 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, 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. Both situations are undesirable. To mitigate this problem, we next describe an algorithm that assists the user 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 similar parameters [Anderberg 1973; Han and Kamber 2000; Jain and Dubes 1988]. For example, partitional clustering methods require the user to specify the *number of clusters* that the data set is supposed to contain. Similarly, density-based clustering 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). Apart from a few ad hoc rules [Anderberg 1973; Jain and Dubes 1988], there exist no guidelines for choosing the operating point.

We now describe the revised scheme for setting the *min_size* parameter. As before, *min_size* is an input parameter that the user has to set. However, the user-defined *min_size* value is no longer used as is, but rather serves as the seed value of an iterative process that converges toward 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, for example, $\varepsilon = 0.05$. 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 result. 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 conviction that it is better to have too many alarm clusters, rather than to mix unrelated alarms in a single cluster.

6. EXPERIENCE WITH ALARM CLUSTERING

This section summarizes our experience with alarm clustering. Section 6.1 investigates the choice of generalization hierarchies. Section 6.2 presents a detailed example to illustrate how alarm clustering and root cause analysis work in practice. Section 6.3 discusses the risk of discarding true positives when filtering rules are derived based on one's understanding of root causes.

6.1 Definition of Generalization Hierarchies

Our alarm-clustering framework assumes that meaningful generalization hierarchies have been defined for all alarm attributes. Figure 1(c) shows that such generalization hierarchies exist for IP addresses and port numbers. This section suggests further generalization hierarchies for numerical, time, and string-valued attributes. In fact, intrusion detection alarms can contain all of these attribute types:

Numerical Attributes. Examples of numerical attributes include counters (e.g., for the number of SYN packets in a SYN flooding alarm) and size fields (e.g., for the packet size in "large ICMP traffic" alarms [CERT 1996]).

Time Attributes. All alarms are time-stamped. However, time should not be treated as a numerical attribute, because doing so would mean to ignore its unique semantics, including notions such as periodicity, workdays versus weekends, and so on.

String Attributes. String attributes assume *arbitrary* and *unpredictable* text values. A typical string attribute is the *context*, which stores the raw audit records or network packets that the IDS believes to constitute and attack. Not all IDSs set the context, but when available, it can significantly facilitate the investigation of alarms. Note that attributes such as "attack name" or "recommended action" are *not* string attributes because they assume values out of a small and predetermined domain.

By defining generalization hierarchies, the user encodes his or her background knowledge about the application domain. Like most knowledge-engineering tasks, there is no single best way to do this. The pros and cons of different generalization hierarchies must be weighted, and a savvy decision must be taken. Therefore, we do not intend to propose the "right" generalization hierarchies. Rather, the generalization hierarchies of this section should be seen as examples that demonstrate the usefulness and versatility of our alarm-clustering framework. Other generalization hierarchies are possible and might prove useful in practice.

6.1.1 *Numerical Attributes.* Generalization hierarchies for numerical attributes are obtained by discretizing the attribute domain into a hierarchically nested sequence of 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 [Dougherty et al. 1995; Han and Fu 1994; Lu 1997].

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 [Han and Fu 1994]. 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 counterintuitive. To mitigate this problem, clustering has been used to find intervals that reflect the natural grouping of the data values [Lu 1997; Miller and Yang 1997]. Other algorithms for the construction of dynamic generalization hierarchies are surveyed by Dougherty et al. [1995]. 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.

6.1.2 *Time Attributes.* 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

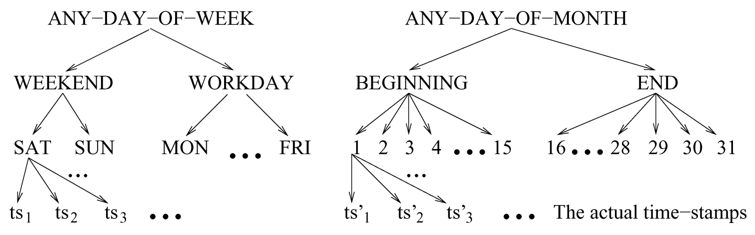


Fig. 3. Sample generalization hierarchies for time attributes.

concepts like these, one can use generalization hierarchies such as the those in Figure 3. 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 both generalization hierarchies of Figure 3 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 the resulting generalization hierarchy does not contain concepts such as "Mon 2nd," which specify a day of the week *plus* a day of the month. To mitigate this problem, one could construct the "full cross-product" of the two generalization hierarchies. The resulting generalization hierarchy contains all concept pairs, including "Mon 2nd." Unfortunately, the generalization hierarchy resulting from a full cross-product is generally complex. It is therefore easier to 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. Because of its simplicity, we use this second option in our experiments.

6.1.3 *String Attributes.* String attributes can assume arbitrary and completely unforseeable text values. Therefore, the challenge lies in tapping the semantic information of these strings. One possible solution to 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, string attributes are another instance where dynamic generalization hierarchies are useful.

A potential problem of this feature-based approach is that it can result in very large generalization hierarchies. Specifically, if the feature extraction step identifies m features, then the complete generalization hierarchy has $2^m - 1$

nodes, namely, one node for each nonempty subset of the feature set. To keep the size of the generalization hierarchy at a manageable level, we use a variant of the above idea in our own experiments. Specifically, let L be an alarm log, let A be a string attribute, and let $V := \langle \mathbf{a}[A] \mid \mathbf{a} \in L \rangle$ be the multiset (a.k.a. bag or collection) of values that attribute A assumes in the alarm log L. We run the Teiresias algorithm [Rigoutsos and Floratos 1998] 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.

6.2 An Illustrative Example

This section describes a clustering experiment that we have conducted on a log of historical alarms. The alarm log is taken from a commercial network-based IDS, spans a time period of one month, and contains 156,380 alarm messages. The IDS sensor was deployed in a network that is isomorphic to the one in Figure 1(a). Please note that this experiment involves *real* alarm data that was collected during the day-to-day operation of a commercially used network. Furthermore, it is worth mentioning that we have applied the same algorithm to many more data sets of various environments. The results obtained with these different data sets are consistent with the results presented here [Julisch and Dacier 2002]. However, this section focuses on a single data set in order to have a detailed discussion of the results.

For the purpose of our experiment, we model alarms as 7-tuples. In detail, the individual alarm attributes are the source and destination IP address, the source and destination port, the alarm type (which encodes the observed attack), the time-stamp, and the context. Please recall that the context is optional, but when present, contains the suspicious network packet. For IP addresses and port numbers, we use the generalization hierarchies in Figure 1(c). For time-stamps, we use the generalization hierarchies in Figure 3. As described at the end of the previous section, we use generalization hierarchies based on frequent substrings for the context. Finally, the alarm-clustering method is configured to enforce ε-robustness (cf. Section 5.3) with $\varepsilon = 0.05$.

Table II shows the generalized alarms of the 13 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 [Tanenbaum 1996]. 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 1(a).

Table II. Generalized Alarms of the 13 Largest Alarm Clusters

Alarm Type	Source-Port	Source-IP	Dest-Port	Dest-IP	Time	Context	Size
WWW IIS view source attack	*NON-PRIV*	*INTERNET*	80	*ip4*	Any	See text	54310
WWW IIS view source attack	*NON-PRIV*	*INTERNET*	80	*ip5*	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	*ip6*	Undefined	*ip1*	*WORKDAY*	Undefined	4581
IP fragment attack	Undefined	*ip6*	Undefined	*ip2*	*WORKDAY*	Undefined	3708
TCP SYN host sweep	*NON-PRIV*	*ip1*	80	any	Any	Undefined	761
TCP SYN host sweep	*NON-PRIV*	*ip2*	80	any	Any	Undefined	663
TCP SYN host sweep	*NON-PRIV*	*FIREWALL*	25	any	Any	Undefined	253
Fragmented ICMP traffic	Undefined	*INTERNET*	Undefined	*ip4*	Any	Undefined	823
Fragmented ICMP traffic	Undefined	*INTERNET*	Undefined	*ip5*	Any	Undefined	711
Unknown protocol field in IP packet	Undefined	*ip7*	Undefined	*FIREWALL*	*END-OF-MONTH*, *TUESDAY*	Undefined	861

Note that the generalized alarms of Table II 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 II. The discussion proceeds by alarm type:

WWW IIS View Source Attack. The first two generalized alarms of Table II 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 II, it has been issued more than 100,000 times. Our analysis has shown that "WWW IIS view source attack" alarms occur when a GET request contains "%2E," as is the case for the one above. The root cause lies in the search engine that the Web servers *ip4* and *ip5* 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 II 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 *ip6* is maliciously sending fragmented packets to the firewalls or there is a router that fragments the packets between *ip6* and the firewalls. Our investigation has shown that the second hypothesis is correct.

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, that is, 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, *ip7* is trying to set up a covert channel.

Note that for the last two alarm types, we found it difficult to pinpoint 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 *ip7*, 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 *ip7* 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 obtain. Specifically, we wrote filtering rules that discarded all alarms matching one of the generalized alarms in Table II. We then applied these filters to the alarms of the following month. The result was that 82% of all alarms were automatically discarded by the filtering rules. Thus, if the root causes had been resolved, then 82% less work would have been the estimated payoff in the subsequent month.

By manually inspecting the 18% of alarms that did not match any filtering rule, we observed that an estimated 30% of these alarms belonged to one of the previously identified root causes. We concluded that our filtering rules were too restrictive and did not match all alarms that the root causes triggered in the second month. As a consequence, we found that 87% is a better estimate for the percentage of alarms that could have been eliminated by removing root causes. It might have been possible to obtain this revised percentage in the first place by using laxer-filtering rules, but that would have entailed a higher risk of discarding unrelated alarms. This is clearly undesirable, in particular, because it also increases the risk of filtering out true positives. This latter risk is further studied in the following section.

6.3 On the Risk of Filtering

Once root causes have been identified, one should devise a strategy for dealing with them in the future. For example, compromised machines, broken component (such as the fragmenting router of the previous section), or configuration faults (which some IDSs confuse with attacks) should be fixed, so that they can no longer trigger alarms in the future. Occasionally, however, root causes are not under our control or they are expensive to fix. Then, custom-made filtering rules, which automatically discard alarms, are an alternative. However, writing filtering rules requires care, or else the risk of discarding true positives can be high. To minimize this risk it is advisable 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 at the beginning of Section 6.1—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. Conversely, when filtering rules are publicly known, then it becomes easier for an attacker to design an attack whose alarms match a filtering rule and get discarded.

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 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 nonvulnerable targets [Lippmann et al. 2002].

More research is needed to quantitatively measure the risk of discarding true positives. In fact, quantifying this risk is similar to measuring the number of false negatives of an IDS. Given that this latter problem is very difficult [McHugh 2000], it seems unlikely that there is a simple way to quantitatively measure the risk that filtering discards true positives.

7. CONCLUSIONS AND FUTURE WORK

This paper has considered the problem of intrusion detection systems overloading their human operators by triggering thousands of alarms per day. Our solution to this problem exploits the observation that in most environments, a few dozens of root causes generally account for over 90% of all alarms. Moreover, these root causes tend to be highly persistent, that is, they do not disappear unless someone removes them. Predominant and persistent root causes are problematic because they cause an alarm flood that distracts the intrusion detection analyst from spotting more subtle attacks. Therefore, we argue that intrusion detection alarms should be handled by identifying and removing the most predominant root causes. To make this idea practical, we introduce alarm clustering as a method that supports the discovery of root causes. Furthermore, we show that the future alarm load decreases significantly if the discovered root causes are removed. Thanks to this reduction in alarm load, it becomes more cost-effective and less error-prone to analyze the remaining intrusion detection alarms.

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

—At present, interpreting alarm clusters in terms of root causes is manual. 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.
—Future work could continue the work of Section 6.3 and develop methods to quantitatively assess the risk of filtering.
—Future work could investigate the effect that different generalization hierarchies have on the clustering results.
—In the IDMEF format [Erlinger and Staniford-Chen], it is possible for alarms to have set-valued attributes. For instance, an alarm may report an attack against several targets. Future work could extend our framework to support such set-valued alarm attributes.

ACKNOWLEDGMENTS

The author thanks Joachim Biskup and Marc Dacier for their valuable comments on earlier versions of this paper. I also thank the anonymous reviewers for their careful and informative suggestions.

This research was supported by the European IST Project MAFTIA (IST-1999-11583), which is partially funded by the European Commission and the Swiss Department for Education and Science. The views herein are those of the author and do not necessarily reflect the views of the supporting agencies.

REFERENCES

AGRAWAL, R., GEHRKE, J., GUNOPULOS, D., AND RAGHAVAN, P. 1998. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD International Conference on Management of Data*, 94–105.

ALLEN, J., CHRISTIE, A., FITHEN, W., MCHUGH, J., PICKEL, J., AND STONER, E. 2000. State of the practice of intrusion detection technologies. Tech. Rep., Carnegie Mellon University. `http://www.cert.org/archive/pdf/99tr028.pdf`.

ALMGREN, M., DEBAR, H., AND DACIER, M. 2000. A lightweight tool for detecting web server attacks. In *Network and Distributed System Security Symposium (NDSS 2000)*, 157–170.

ANDERBERG, M. R. 1973. *Cluster Analysis for Applications*. Academic Press, New York.

AXELSSON, S. 2000. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security 3*, 3, 186–205.

BACE, R. 2000. *Intrusion Detection*. Macmillan Technical Publishing.

BARBARÁ, D. AND JAJODIA, S., (Eds.). 2002. *Applications of Data Mining in Computer Security*. Kluwer Academic Publishers, Boston, MA.

BARBARÁ, D., WU, N., AND JAJODIA, S. 2001. Detecting Novel network intrusions using Bayes estimators. In *1st SIAM International Conference on Data Mining (SDM'01)*.

BELLOVIN, S. M. 1993. Packets found on an Internet. *Computer Communications Review 23*, 3, 26–31.

BLOEDORN, E., HILL, B., CHRISTIANSEN, A., SKORUPKA, C., TALBOOT, L., AND TIVEL, J. 2000. Data mining for improving intrusion detection. `http://www.mitre.org/support/papers/tech_papers99_00/`.

BOULOUTAS, A., CALO, S., AND FINKEL, A. 1994. Alarm correlation and fault identification in communication networks. *IEEE Transactions on Communications 42*, 2–4, 523–533.

BRODERICK, J. (ed.) 1998. IBM outsourced solution. `http://www.infoworld.com/cgi-bin/displayTC.pl?/980504sb3-ibm.htm`.

CERT. 1996. Advisory CA-1996-26: Denial-of-service attack via ping. `http://www.cert.org/advisories/CA-1996-26.html`.

CERT. 2001. Advisory CA-2001-19: "Code Red" worm exploiting buffer overflow in IIS indexing service DLL. `http://www.cert.org/advisories/CA-2001-19.html`.

CHAN, P. AND STOLFO, S. 1998. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *4th International Conference on Knowledge Discovery and Data Mining*, 164–168.

CHEUNG, D., HWANG, H., FU, A., AND HAN, J. 2000. Efficient rule-based attribute-oriented induction for data mining. *Journal of Intelligent Information Systems 15*, 2, 175–200.

CLIFTON, C. AND GENGO, G. 2000. Developing custom intrusion detection filters using data mining. In *Military Communications International Symposium (MILCOM2000)*.

CUPPENS, F. 2001. Managing alerts in a multi-intrusion detection environment. In *17th Annual Computer Security Applications Conference (ACSAC)*, 22–31.

CUPPENS, F. AND MIÈGE, A. 2002. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy, Oakland, CA*.

DAIN, O. AND CUNNINGHAM, R. K. 2002. Fusing heterogeneous alert streams into scenarios. See Barbará and Jajodia [2002].

DEBAR, H., DACIER, M., AND WESPI, A. 2000. A revised taxonomy for intrusion detection systems. *Annales des Télécommunications 55*, 7–8, 361–378.

DEBAR, H. AND WESPI, A. 2001. Aggregation and correlation of intrusion-detection alerts. In *4th Workshop on Recent Advances in Intrusion Detection (RAID)*. LNCS. Springer Verlag, Berlin, 85–103.

DOUGHERTY, J., KOHAVI, R., AND SAHAMI, M. 1995. Supervised and unsupervised discretization of continuous features. In *Proceedings of the 12th International Conference on Machine Learning*, 194–202.

ERLINGER, M. AND STANIFORD-CHEN, S. Intrusion detection exchange format (IDWG). `http://www.ietf.org/html.charters/idwg-charter.html`.

FAWCETT, T. AND PROVOST, F. 1997. Adaptive fraud detection. *Data Mining and Knowledge Discovery 1*, 291–316.

HAN, J., CAI, Y., AND CERCONE, N. 1992. Knowledge discovery in databases: An attribute-oriented approach. In *18th International Conference on Very Large Databases*, 547–559.

HAN, J., CAI, Y., AND CERCONE, N. 1993. Data-driven discovery of quantitative rules in relational databases. *IEEE Transactions on Knowledge and Data Engineering 5*, 1, 29–40.

HAN, J. AND FU, Y. 1994. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, 157–168.

HAN, J. AND KAMBER, M. 2000. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publisher, San Mateo, CA.

HELLERSTEIN, J. L. AND MA, S. 2000. Mining event data for actionable patterns. In *The Computer Measurement Group*.

HOUCK, K., CALO, S., AND FINKEL, A. 1995. Towards a practical alarm correlation system. In *4th International Symposium on Integrated Network Management*. A. S. Sethi, Y. Raynaud, and F. Faure-Vincent, Eds. Chapman & Hall, London, 226–237.

ILUNG, K. 1993. USTAT: A real-time intrusion detection system for UNIX. In *IEEE Symposium on Security and Privacy*, Oakland, CA, 16–28.

JAIN, A. AND DUBES, R. 1988. *Algorithms for Clustering Data*. Prentice-Hall, Englewood Cliffs, NJ.

JAKOBSON, G. AND WEISSMAN, M. D. 1993. Alarm Correlation. *IEEE Network 7*, 6, 52–59.

JAKOBSON, G. AND WEISSMAN, M. D. 1995. Real-time telecommunication network management: Extending event correlation with temporal constraints. In *4th International Symposium on*

Integrated Network Management. A. S. Sethi, Y. Raynaud, and F. Faure-Vincent, Eds. Chapman & Hall, London, 290–301.

JULISCH, K. 2001. Mining alarm clusters to improve alarm handling efficiency. In *17th Annual Computer Security Applications Conference (ACSAC)*, 12–21.

JULISCH, K. AND DACIER, M. 2002. Mining intrusion detection alarms for actionable knowledge. In *8th ACM International Conference on Knowledge Discovery and Data Mining*, 366–375.

KLEMETTINEN, M. 1999. A knowledge discovery methodology for telecommunication network alarm data. Ph.D. Thesis, University of Helsinky, Finland.

KUMAR, S. 1995. Classification and detection of computer intrusions. Ph.D. Thesis, Purdue University.

LAPRIE, J., Ed. 1992. *Dependability: Basic Concepts and Terminology*. Dependable Computing and Fault-Tolerant Systems, vol. 5. Springer-Verlag, Vienna.

LEE, W. AND STOLFO, S. J. 2000. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security 3*, 4, 227–261.

LEWIS, L. 1993. A case-based reasoning approach to the resolution of faults in communication networks. In *3th International Symposium on Integrated Network Management*. H.-G. Hegering and Y. Yemini, Eds. North Holland, Amsterdam, 671–682.

LIN, D. 1998. An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, 296–304.

LIPPMANN, R., WEBSTER, S., AND STETSON, D. 2002. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *5th Workshop on Recent Advances in Intrusion Detection (RAID)*. LNCS, vol. 2516. Springer Berlin, Verlag, 307–326.

LU, Y. 1997. Concept hierarchy in data mining: Specification, generation, and implementation. M.S. Thesis, Simon Fraser University, Canada.

MANGANARIS, S., CHRISTENSEN, M., ZERKLE, D., AND HERMIZ, K. 2000. A data mining analysis of RTID alarms. *Computer Networks 34*, 4, 571–577.

MCHUGH, J. 2000. 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 3*, 4, 262–294.

MILLER, R. AND YANG, Y. 1997. Association rules over interval data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 452–461.

MILLIGAN, G. W. 1996. Clustering validation: Results and implications for applied analyses. World Scientific Publishing, Singapore, 341–375.

MOUNJI, A. 1997. Languages and tools for rule-based distributed intrusion detection. Ph.D. Thesis, Facultes Universitaires Notre-Dame de la Paix Namur, Belgium.

NING, P., JAJODIA, S., AND WANG, X. 2001. Abstraction-based intrusion detection in distributed environments. *ACM Transactions on Information and System Security 4*, 4, 407–452.

NYGATE, Y. A. 1995. Event correlation using rule and object based techniques. In *4th International Symposium on Integrated Network Management*. A. S. Sethi, Y. Raynaud, and F. Faure-Vincent, Eds. Chapman & Hall, London, 278–289.

OHSIE, D. A. 1998. Modeled abductive inference for event management and correlation. Ph.D. Thesis, Columbia University.

PAPADIMITRIOU, C. H. 1994. *Computational Complexity*. Addison-Wesley, Reading, MA.

PAXSON, V. 1999. Bro: A system for detecting network intruders in real-time. *Computer Networks 31*, 23/24, 2435–2463.

PENG, Y. AND REGGIA, J. A. 1987a. A probabilistic causal model for diagnostic problem solving—Part I: Diagnostic strategy. *IEEE Transactions on Syst. Man Cybern. 17*, 3, 395–404.

PENG, Y. AND REGGIA, J. A. 1987b. A probabilistic causal model for diagnostic problem solving—Part I: Integrating symbolic causal inference with numeric probabilistic inference. *IEEE Trans. Syst. Man. Cybern. 17*, 2, 146–162.

POWELL, D. AND STROUD, R. 2001. Architecture and Revised Model of MAFTIA. Tech. Rep. CS-TR-749, University of Newcastle upon Tyne.

PRICE, K. E. 1997. Host-based misuse detection and conventional operating systems' audit data collection. M.S. Thesis, Purdue University.

PTACEK, T. H. AND NEWSHAM, T. N. 1998. Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. Rep., Secure Networks, Inc.

Rada, R. and Bicknell, E. 1989. Ranking documents with a thesaurus. *Journal of the American Society for Information Science 40*, 5, 304–310.

Rada, R., Mill, H., Bicknell, E., and Blettner, M. 1989. Development and application of a metric on semantic nets. *IEEE Transactions on Syst. Man Cybern. 19*, 1, 17–30.

Resnik, P. 1999. 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.

Rigoutsos, I. and Floratos, A. 1998. Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm. *Bioinformatics 14*, 1, 55–67.

Sekar, R., Guang, Y., Verma, S., and Shanbhag, T. 1999. A high-performance network intrusion detection system. In *6th ACM Conference on Computer and Communications Security*, 8–17.

Staniford, S., Hoagland, J. A., and McAlerney, J. M. 2000. Practical automated detection of stealthy portscans. In *ACM Computer and Communications Security IDS Workshop*, 1–7.

Tanenbaum, A. S. 1996. *Computer Networks*. Prentice-Hall, Englewood Cliffs, NJ.

Valdes, A. and Skinner, K. 2001. Probabilistic alert correlation. In *4th Workshop on Recent Advances in Intrusion Detection (RAID)*. LNCS. Springer Verlag, Berlin, 54–68.

Yemini, S., Kliger, S., Mozes, E., Yemini, Y., and Ohsie, D. 1996. High speed & robust event correlation. *IEEE Communications Magazine 34*, 5, 82–90.

Zamboni, D. 2001. Using internal sensors for computer intrusion detection. Ph.D. Thesis, Purdue University.

Received May 2002; revised January 2003, July 2003; accepted July 2003