Alarm Clustering for Intrusion Detection Systems in Computer Networks

Giorgio Giacinto, Roberto Perdisci, Fabio Roli

Department of Electrical and Electronic Engineering, University of Cagliari
Piazza D Armi - 09123 Cagliari, Italy
{giacinto,roberto.perdisci,roli}@diee.unica.it

Abstract. Until recently, network administrators manually arranged alarms produced by Intrusion Detection Systems (IDSs) to attain a high-level description of threats. As the number of alarms is increasingly growing, automatic tools for alarm clustering have been proposed to provide such a high level description of the attack scenario. In addition, it has been shown that effective threat analysis require the *fusion* of different sources of information, such as different IDSs, firewall logs, etc. In this paper, we propose a new strategy to perform alarm clustering which produces unified descriptions of attacks from multiple alarms. Tests have been performed on a live network where commercial and open-source IDSs analyzed network traffic.

Keywords: Computer Security, Intrusion detection, Clustering

1 Introduction

At present, a number of commercial, open-source, and research Intrusion Detection Systems (IDSs) tools are available. They differ in the way intrusions are detected, and in the available options allowing further alarm processing. Among them, network misuse detectors are widely used in many organizations for their ability in detecting well-known patterns of intrusions.

Network misuse detectors analyse network traffic looking for packets whose characteristics match the "signature" of known attacks. As soon as a signature is matched, an alarm is raised. As signature matching is performed on a packet basis, alarms provide a powerful source of fine-grain information related to suspect activities in the protected network. In order to gain an understanding of the intrusions against the protected network, a network administrator needs to arrange these alarms to produce a high-level description of the threat. As the number of alarms is increasingly growing, it is not feasible for a network administrator to manually arrange the huge volume of alarms. Recently a number of alarm clustering products have been proposed to provide such a high level description of the attack scenario [1]. The aim is to manage the large number of so-called *elementary* alarms produced by IDSs, by their fusion in higher-level alarm messages. The source of such a large number of alarms is motivated by the nature of some categories of attacks which send a large number of malicious packets. As signature-based IDSs produce an alarm for each malicious packet, alarm flooding may occur. Alarm clustering can also be used to *fuse* alarms from different sensors. The use of multiple *complementary* Intrusion Detection technologies can provide the following benefits: i) for a given attack, different IDSs may produce different outputs; ii) for a given attack, only a limited number of IDSs might be able to detect it; iii) the fusion of alarms raised by different IDSs may attain more comprehensive information about intrusion attempts than that attained using a single IDS technique. Therefore, the proposed multiple-sensor environment is made up of a number of IDSs (e.g., commercial and open-source products), and the measurements to be fused are the elementary alarms raised by each IDS. This paper proposes an *on-line* alarm clustering algorithm whose output is a set of *meta-alarms*. During the operation of the IDSs, the alarms are analysed and clustered. When no further alarm can be clustered to an existing group, the related *meta-alarm* is output to the administrator.

Meta-alarms provide a network administrator with summary information about the attack and the related alarm messages produced by IDSs. This information can be further used by higher-level modules that perform *multiple-step attack* scenario reconstruction and threat analysis.

At present, a few works on alarm clustering and correlation have been presented [2-5]. With respect to the related work, in the present paper a novel *on-line* alarm-clustering algorithm is proposed. The objective is to achieve alarm volume reduction by fusing alarms produced by different sensors in consequence of a given attack. In particular, the main contribution is the introduction of a learning phase, which aims at extracting the attack class(es) an attack description belongs to. This attack description classification process allows to cluster alarms seemingly produced by different attacks but belonging to the same alarm thread.

The paper is organized as follows. Section 2 presents the details of the proposed alarm clustering algorithm. Some results attained on a test network with commercial and open-source IDSs are reported in Section 3. In particular, the structure of the meta-alarm is presented which can summarize a large number of elementary alarms. Conclusions are drawn in Section 4.

2 The Proposed Alarm Clustering Algorithm

In this section, we present our alarm clustering algorithm designed to process the sequence of alarms produced by IDSs, and then produce *meta-alarms*, i.e. summary descriptions of events obtained by aggregating correlated alarms produced by various IDS sensors. Such a summary information can be provided by the attack class the alarms refer to. The alarm class provides an effective high-level information [6] that can be used by higher-level modules that perform multiple-step attack scenario reconstruction. As an example, let us consider the three attack classes used in our experiments, i.e., *portscan*, web-*scan*, and *DoS* (Denial of Service). A *portscan* attack is performed by sending a very large number of TCP or UDP packets to different ports in order to spot whether a

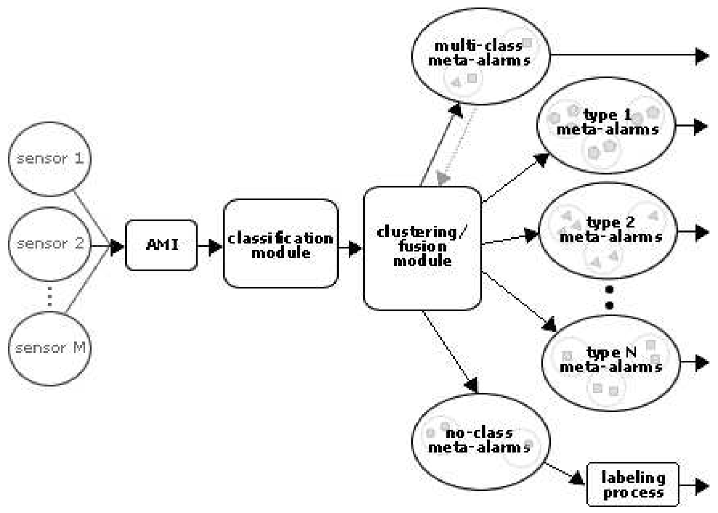

Fig. 1. Alarm Clustering Module

service is bound to a certain port or not. In a similar way a *webscan* attack is performed by sending a sequence of HTTP queries to a web server (the victim) looking for vulnerable web applications. DoS attacks, instead, are usually performed by sending a large number of properly crafted packets to a victim host trying to exploit vulnerabilities which can cause the victim host or application to crash. Each meta-alarm is further described by a number of features needed to uniquely identify the event, such as start and stop times, source and destination IP, etc. In addition, the identifiers of the aggregated alarm logs are reported for further inspection. The proposed system results particularly suitable in aggregating alarms produced by those kinds of attacks which cause the IDSs to produce a high number of alarms. In the following, we will refer primarily to signature-based Netowrk-IDS (NIDS) sensors, as it is the most widely used type of IDS sensors. Nevertheless, the reported discussion can be extended to other ID techniques. We will provide an overview of the architecture of the proposed alarm-clustering module first, then going into the details of each of the components. Figure 1 depicts a schema of our alarm clustering system.

The first block is the Alarm Management Interface (AMI) that performs data alignment by translating each alarm message toward a standard alarm message format. This is necessary because IDSs from different vendors usually produce messages according to different formats. In our implementation, we used the IDMEF format because it has been proposed as standard format by the IETF [7]. The second block, i.e. the *classification module*, is designed to label an alarm message as belonging to one or more attack classes. The *classification module* is motivated by two kinds of ambiguities: i) for a given attack, different sensors may produce a number of alarms reporting different attack descriptions; ii) an attack description may be produced by the IDS in response to different attacks. In case of alarms labelled as belonging to more than one class, the clustering process removes the ambiguity as soon as an alarm with a unique label is clustered with the multiple-class alarm. In fact, each cluster must contain alarms belonging to one attack class. We also used the ”no-class” label for those meta-alarms related to attacks for which a class has not been defined during classifier design. Details on the *training* procedure of the classification module will be given in Section 2.2. Classified alarms are sequentially sent to the *clustering/fusion* block, which applies a nearest-neighbour clustering algorithm [8]. For each received alarm the clustering algorithm determines whether the alarm can be clustered, and thus fused, to one of the clusters or have to initialize a new *meta-alarm* (a new group of alarms). The whole system is designed to be used in a near real-time environment, i.e. IDS sensors send the alarms to the alarm reduction system as soon as they are produced. In such an environment meta-alarms that have not been involved in a *clustering/fusion* process for a time interval greater than a predefined timeout threshold will be removed from the reduction system and sent in IDMEF format to the administrator.

2.1 Meta-alarm content

Before going into the details of the clustering algorithm, let us state which information we aim to extract from each cluster, and include in the related *meta-alarm.* A *meta-alarm* is characterised by the following features: a classification name, that is the common generalized class-name assigned to the clustered alarms by the classification module; the *create-time*, that is the timestamp of the last meta-alarm update (the last fusion). Additional data include the start and stop times of the attack, list of source IP addresses, list of target IP addresses, source ports, target ports, etc. In addition, a reference to the log files of the IDSs is reported so that further investigation of the aggregated alarms can be carried out. It is worth noting that a meta-alarm M is removed from the clustering-fusion module and reported to the administrator if no more alarms are fused within a suitable time threshold.

2.2 Classification module

An alarm class C is made up of the set of alarm messages provided by the sensors in response to at tacks of type C. For example, the *portscan* alarm class is made up of the set of alarm messages obtained by simulating *portscan* attacks with different techniques. We have already noticed that a given alarm can be raised by an IDS in response to different attacks. For example, a *portscan* may cause an IDS to produce a number of alarms that refer to *DoS* attacks in addition to the alarms related to the *portscan* activity. Such *DoS* alarms are confusing because they should be produced only in case of real *DoS* attacks. The role of the classification module is to assign each alarm to the attack class(es) that might have produced it. To this end, the classifier is designed by simulating a number of real attacks for each class of attacks. For example, if we consider attacks belonging to *portscan*, *webscan* and *DoS* classes, the designing process has to be performed in the following way:

1. Simulate the most frequent *portscan* attacks with different techniques.
2. Extract the pairs {sensor-name, alarm-message} from each alarm produced by step 1.
3. Store the pairs {sensor-name, alarm-message} into a set called *portscan-descriptions*.
4. Repeat steps 1, 2 and 3 for *webscan* and *DoS* attacks, thus storing the pairs {sensor-name, alarm-message} into *webscan-descriptions* set and *dos-descriptions* set respectively.

When the classifier receives an alarm with description *Desc-1* produced by *Sensor-A*, the pair {*Sensor-A*, *Desc-1*} is compared to each pair contained into the *portscan-descriptions*, *webscan-descriptions* and *dos-descriptions* sets. The alarm is then labelled with the classes with matching pairs. For example, if the pair {*Sensor-A*, *Desc-1*} is found both into the *portscan-descriptions* set, and into the *dos-description* set, then the alarm will be labelled as belonging to both portscan and *DoS* classes. On the other hand, if the pair {*Sensor-A*, *Desc-1*} is not in any sets of descriptions, the alarm will be labelled as "no-class".

2.3 Clustering/fusion module

The clustering/fusion module is initialised by creating a number of empty sets, each one devoted to contain clusters related to one of the attack classes taken into account in the classifier design phase. In the case of the above example the clustering/fusion block creates three sets: *PortscanMetaAlarmSet*, *Webscan-MetaAlarmSet*, and *DoSMetaAlarmSet*. In addition, two other sets are created, namely the *MultiClassMetaAlarmSet*, and the *NoClassMetaAlarmSet*. The first set is devoted to temporarily contain meta-alarms obtained by clustering alarms labelled as belonging to multiple classes, while the latter is devoted to contain meta-alarms obtained by clustering alarms that have not received a label from the classification module. It is worth recalling that alarm clustering is aimed at reducing the number of alarms produced by a certain class of attacks. Thus, a number of alarms are clearly not taken into account in the design phase. Before giving the details of the clustering algorithm, some definitions have to be introduced:

- **Definition 1.** *Distance between pairs of features*
  Let us denote the distance between the i-th feature of an alarm A and the corresponding feature of a meta-alarm M as $dist(A.feat_i, M.feat_i)$. *Distance measures for various types of features such as the timestamp, target IP, target port, source IP, source port, etc., are defined in Section 2.4.*
- **Definition 2.** *Clustering function*
  An alarm A is assigned to the nearest cluster if the distance between A and the meta-alarm M associated with that cluster is below a predefined threshold. In particular, alarm A is assigned to the nearest cluster M if all the

distances between the corresponding features are smaller than a set of predefined thresholds:

$$dist(A.feat_i, M.feat_i) \leq thres_i \qquad \forall i = 1, .., v \qquad (1)$$

where v is the total number of features. The values of the thresholds $\{thres_i\}_{i=1..v}$ depend on the class M belongs to, as well as on the characteristics of the protected network. In the following, we will refer to an alarm A and a meta-alarm M satisfying Eq. 1, to be correlated.

– **Definition 3.** *Distance between an alarm and a meta-alarm*
If an alarm A and a meta-alarm M are correlated, then their distance is computed as the time gap between the create-time of A and the create-time of the more recent alarm fused to M. Otherwise, the distance between A and M is set to $+\infty$.

– **Definition 4.** *Distance between an alarm and a meta-alarm set*
The distance between an alarm A and a meta-alarm set S is defined in the following way:
1. *If S does not contain any meta-alarm M correlated to A, then the distance is set to $+\infty$.*
2. *If S contains k meta-alarms $M_1, M_2, ..., M_k$ correlated to A, the distance between A and S is computed as $\min_{i=1..k}(dist(A, M_i))$.*

In order to explain how the proposed clustering algorithm works, let us resort to an example. Let us suppose to be in a running state, and that each meta-alarm set contains a number of clusters. When a new alarm A is processed by the clustering module, three different cases may occur:

a) A has been labelled as belonging to a unique class.
If the alarm A has been labelled, for example, as a *portscan* the following distances will be computed:
$d_1 = dist(A, PortscanMetaAlarmSet)$
$d_2 = dist(A, MultiClassMetaAlarmSet)$
$d_3 = dist(A, NoClassMetaAlarmSet)$
If ($d_1 = d_2 = d_3 = +\infty$), then there is no meta-alarm correlated to A into the *Portscan*, *MultiClass*, and *NoClass* meta-alarm sets. In this case, A will be inserted into the *PortscanMetaAlarmSet* where it will initialize a new meta-alarm. If $d_1 = min\{d_1, d_2, d_3\}$, A will be inserted into the *Portscan-MetaAlarmSet*, and it will be fused with the nearest portscan meta-alarm that is correlated to A. Similarly, if d_2 or d_3 exhibit the minimum distance, A will be inserted respectively into the *MultiClassMetaAlarmSet* or the *No-ClassMetaAlarmSet*, and it will be fused with the nearest correlated meta-alarm. In the case of $d_2 = min\{d_1, d_2, d_3\}$, the resulting meta-alarm will be moved from the *MultiClassMetaAlarmSet* to the *PortscanMetaAlarmSet*, as the alarm A has a unique class label that can resolve the ambiguity of the correlated meta-alarm. In the case of $d_3 = min\{d_1, d_2, d_3\}$, the class label given to A will not be further considered, and the resulting meta-alarm will have no class label. The reason for computing the distances d_2 and d_3

instead of immediately insert A into *PortscanMetaAlarmSet* (A has been labeled as a *portscan* by the classification module) is justified by the following considerations: 1) Let us assume that alarm A is the n-th alarm caused by a *portscan* attack, and that the first $n-1$ alarms have been classified as belonging to multiple classes, *portscan* class included. By comparing A with the meta-alarms contained in the *MultiClassMetaAlarmSet*, it will be correctly fused with the correct sequence of alarms. 2) Given that a perfect matching is required among the features of the alarm A and those of a no-class meta-alarm M to be correlated (Eq. 1), if $d_3 = min\{d_1, d_2, d_3\}$, A and M are quite certainly related to the same attack even though A has been labelled as belonging to a certain class.

b) A has been labelled as belonging to multiple classes.
If alarm A has been labelled, e.g. as *portscan* and *DoS*, the following four distances will be computed:
$d_1 = dist(A, PortscanMetaAlarmSet)$
$d2 = dist(A, DosMetaAlarmSet)$
$d3 = dist(A, MultiClassMetaAlarmSet)$
$d4 = dist(A, NoClassMetaAlarmSet)$
if $(d_1 = d_2 = d_3 = d_4 = +\infty)$, A will be inserted into the *MultiClassMetaAlarmSet*, and it will initialize a new meta-alarm. If one or more distances are not equal to $+\infty$, then A will be inserted into the nearest meta-alarm set, and it will be fused with the nearest meta-alarm.

c) A has been labelled as belonging to none of the classes.
If A has been labelled as belonging to no-class, then it will be inserted into the *NoClassMetaAlarmSet*, and it will be clustered with the nearest no-class meta-alarm. If the *NoClassMetaAlarmSet* contains no meta-alarm correlated to A, then A will initialize a new no-class meta-alarm that inherits A's features. It is worth recalling that an alarm A and a no-class meta-alarm M are considered correlated only if all A's and M's features (except the attack description) perfectly match. In this case there is a high probability that A and M are relative to the same attack, even if the attack descriptions do not coincide.

2.4 Distances among features

In this section we present the definition of some of the distances among features used by the clustering algorithm. Let A be an alarm and M a meta-alarm. Distances among IP addresses and port lists hold the same definitions either they refer to target or source information (i.e. $dist(A.sourceIP, M.sourceIP)$ and $dist(A.targetIP, M.targetIP)$ have the same definition, as well as distances among source or target port lists).

- $dist(A.IP, M.IP)$: We consider only IPv4 addresses. The distance is defined as a sub-network distance. We take the binary expression of $A.IP$ and $M.IP$, then we XOR the two binary strings. If we call n the number of zeros in the resulting binary string counted starting from the right, the distance d will be $d = 32 - n$. The greater d, the greater the distance between IP addresses.

– $dist(A.portList, M.portList)$: The distance among $A.portList$ and $M.portList$ equals the number of port numbers present in $A.portList$ but not in $M.portList$.
– $time\_distance(A, M)$: The time distance t among an alarm A and a meta-alarm M is computed as the distance, in terms of milliseconds, among $A.createTime$ and $M.stopTime$.

3 Experiments

The proposed alarm clustering strategy has been implemented and tested on a live network containing dozens of hosts, some of them chosen as victims. It is worth noting that at present no dataset is publicly available for designing and testing alarm clustering algorithms. As a consequence, a direct comparison with results in the literature is rather difficult. Thus, researchers usually evaluate their algorithms by performing some experiments on a typical network scenario, and assessing the effectiveness of the proposed techniques on such a scenario. The traffic of the considered network was made up of the so-called background traffic, i.e., the normal activity of the users of the network, and by a number of simulated attacks. Three IDSs have been used to monitor network traffic: Snort 2.1.0 [9], Prelude-NIDS 0.8.6 [10], and ISS Real Secure Network Sensor 7.0 [11]. We have subdivided the experiments into three stages: 1) Training of the classification module; 2) Tuning of the thresholds involved in the clustering algorithm; 3) Performance tests. The first two stages have been carried out by executing attack simulations in an isolated network made up of three hosts, two victims (a Linux host and a Win2k host), and an attacker host. The performed experiments were related to three attack classes, i.e., *portscan*, *webscan*, and *DoS*, as they usually produce a large number of alarms.

3.1 Training and Tuning

The classification module has been designed using a number of tools available in the Internet. In particular, we have used *nmap*, *wups*, etc., as *portscan* tools; *nikto*, *babelweb*, etc., as *webscan* tools; *teardrop*, *jolt* (aka ping of death), *synflood*, etc., as *DoS* attacks. During this phase, for each attack, the pairs {*sensor-name*, *alarm-message*} has been stored according to the procedure described in section 2.2. The values of the thresholds used by the clustering algorithm described in Section 3.3 have been estimated in two phases. In the first phase, an initial value for the thresholds has been chosen by heuristics based on attack characteristics. Then, in the second phase, attack simulations have been performed in the isolated network to suitably tune the thresholds in order to effectively cluster all the correlated alarms produced by a given attack. The notion of effectiveness may change according to the characteristic of the protected network, the needs of the network administrator, etc. Thus different tunings may fit different administrator's needs. The thresholds used in our experiments are reported in Tab.1. The δt constant in the Time threshold column accounts for possible drifts among IDS sensors' clock. In our experiments, δt was set equal to one second.

Meta-Alarm Class	**SourceIP**	**TargetIP**	**SourcePort**	**TargetPort**	**Time**(s)
portscan	0	0	$+\infty$	$+\infty$	480+δt
webscan	0	0	$+\infty$	0	120+δt
DoS	$+\infty$	0	$+\infty$	$+\infty$	120+δt
no-class	0	0	0	0	0+δt

Table 1. Values of the thresholds used in the clustering algorithm

3.2 Performance Tests

A large number of attacks have been executed in the selected live network to test the feasibility of the designed system to correctly cluster attacks. Results showed that the proposed technique produced not only meta-alarms related to the simulated attacks, but also meta-alarms related to the background traffic. As the design phase has been carried out in an isolated network, this results show the feasibility of the proposed approach. Table 2 reports the details of the most significant results.

Portscan - When porstcans have been performed, the clustering algorithm successfully produced a meta-alarm for every portscan activity. As an example, consecutive SYN and Xmas portscans have been performed from one source towards a victim, producing a total of 3074 alarms from the three considered IDSs (see the first column in table 2). During these attacks, the sensors also produced alarms related to malicious activities in the background traffic. The clustering algorithm correctly produced one meta-alarm related to the portscan, and 15 meta-alarms related to other activities. The meta-alarm related to portscan activities is correctly labelled as portscan, and contains the list of scanned ports, the source and target hosts, the start and stop times of the attack, and references to the alarms that originated the meta-alarm.

Webscans - Similar results have been also attained with webscans. In some cases, long attacks originated more than one meta-alarm, because of time gaps among groups of alarms. This kind of anomalies can be resolved by a post-processing situation refinment module that is aimed at finding relationships among meta-alarms. As an example, Webscan2 (nikto) originated 19164 alarms that were clustered into 37 clusters. The size of the first two clusters was equal to 7464 and 11631 alarms, respectively. It is worth noting that 69 alarms produced by the ISS Real Secure sensor generated 35 meta-alarms. These alarms were related to attack responses produced by the webserver that were not correctly recognized by Real Secure.

DoS - Four DoS attacks have been performed against the same host. The proposed alarm clustering was able to correctly produce 4 meta-alarms corresponding to the different attacks carried out and 33 meta-alarms corresponding to alarms related to suspicious background-traffic.

Attack Type	**portscan**	**webscan1**	**webscan2**	**DoS**
Alarms from Snort	1058	94	7143	71
Alarms from Prelude	1314	93	6601	8828
Alarms from ISS Real Secure	728	63	5586	186
Total Number of Alarms	*3100*	*250*	*19330*	*9085*
Attack-related Alarms	**3074**	**244**	**19164**	**9028**
Alarms produced by backgroud traffic	26	6	166	57
Meta-Alarms from simulated attacks	**1**	**1**	**37**	**4**
Meta-Alarms from backgroud traffic	15	3	85	33
Total number of Meta-Alarms	*16*	*4*	*122*	*37*

Table 2. Experimental results on live network

4 Conclusions

In this paper we proposed a novel on-line alarm-clustering algorithm whose main objective is the reduction of the volume of alarms produced by today's IDS sensors. The clustering algorithm has been devised to work in near real time. Experiments performed in different attack scenarios on a live network showed that the proposed algorithm effectively groups alarms related to the same attack, even though IDSs produced alarms whose descriptions were erroneously referred to different types of attacks. The produced meta-alarms provide the system administrator with a concise high-level description of the attack. In addition, it is the starting point for the development of modules for situation refinement and threat analysis.

References

1. J. Haines, D. K. Ryder, L. Tinnel, S. Taylor, *Validation of Sensor Alert Correlators*, IEEE Security Privacy, January-February 2003, 1(1), pp. 46-56.
2. A. Valdes, K. Skinner, *Probabilistic Alert Correlation*, RAID 2001. LNCS 2212, pp. 54-68.
3. F. Cuppens, *Managing Alerts in a Multi-Intrusion Detection Environment*, Proceedings of ACSAC'01, IEEE Computer Society.
4. F. Cuppens, A. Mige, *Alert Correlation in a Cooperative Intrusion Detection Framework*, Proceedings of the IEEE Symposium on Security and Privacy, 2002.
5. P. A. Porras, M. W. Fong, A. Valdes, *A Mission-Impact-Based Approach to INFOSEC Alarm Correlation*, RAID 2002. Springer-Verlag, LNCS 2516, pp. 95-114.
6. J. Undercoffer, A. Joshi, J. Pinkston, *Modeling Computer Attacks: An Ontology for Intrusion Detection*, RAID 2003. Springer-Verlag, LNCS 2820, pp. 113-135.
7. D. Curry, H. Debar, B. Feinstein, *The Intrusion Detection Message Exchange Format* (http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-11.txt)
8. A.K. Jain, M.N. Murty, P.J. Flynn, *Data clustering: a review*, ACM Computing Surveys 31(3) 1999, 264-323.
9. *Snort, Lightweight Intrusion Detection for Networks.* (http://www.snort.org)
10. *Prelude Intrusion Detection System.* (http://www.prelude-ids.org)
11. *ISS, Inc.: RealSecure intrusion detection system.* (http://www.iss.net)