INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt (2011)
Published online in Wiley Online Library (wileyonlinelibrary.com) DOI: 10.1002/nem.804

FuzMet: a fuzzy-logic based alert prioritization engine for intrusion detection systems

Khalid Alsubhi[1,*†], Issam Aib[2] and Raouf Boutaba[1,3]

[1]*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada*
[2]*Softray Business Solutions*
[3]*Division of IT Convergence Engineering, POSTECH, Pohang, KB 790–784, Korea*

SUMMARY

Intrusion detection systems (IDSs) are designed to monitor a networked environment and generate alerts whenever abnormal activities are detected. The number of these alerts can be very large, making their evaluation by security analysts a difficult task. Management is complicated by the need to configure the different components of alert evaluation systems. In addition, IDS alert management techniques, such as clustering and correlation, suffer from involving unrelated alerts in their processes and consequently provide results that are inaccurate and difficult to manage. Thus the tuning of an IDS alert management system in order to provide optimal results remains a major challenge, which is further complicated by the large spectrum of potential attacks the system can be subject to. This paper considers the specification and configuration issues of FuzMet, a novel IDS alert management system which employs several metrics and a fuzzy-logic based approach for scoring and prioritizing alerts. In addition, it features an alert rescoring technique that leads to a further reduction in the number of alerts. Comparative results between SNORT scores and FuzMet alert prioritization onto a real attack dataset are presented, along with a simulation-based investigation of the optimal configuration of FuzMet. The results prove the enhanced intrusion detection accuracy brought by our system. Copyright © 2011 John Wiley & Sons, Ltd.

Received 6 September 2010; Revised 27 July 2011; Accepted 31 July 2011

1. INTRODUCTION

Network attacks are growing more serious, forcing system defenders to deploy more sophisticated security software and devices including firewalls, antivirus tools, intrusion detection systems (IDSs), and information protection systems (IPSs). An IDS can be host based or network based and is aimed to inspect user and/or network activity looking for suspicious behavior and report it to security analysts in the form of alerts. A *signature-based* IDS generates an alert when the traffic contains a pattern that matches signatures of malicious or suspicious activities. An *anomaly-based* IDS examines ongoing activity and detects attacks based on the degree of variation from normal past behavior. However, both mechanismssuffer from the large number of alerts that they generate. These alerts need to be evaluated by security analysts before any further investigation in order to take appropriate action against the attacks.

IDSs usually generate a large number of alerts whenever abnormal activities are detected. Inspecting and investigating all reported alerts manually is a difficult, error-prone, and time-consuming task. In addition, ignoring alerts may lead to successful attacks. To tackle this problem, low-level and high-level alert management processes have been introduced. Low-level alert processing deals with each alert individually to enrich its attributes or assign a score to it based on the potential threat it describes. High-level alert management techniques, such as aggregation, clustering, correlation, and fusion, were

*Correspondence to: Khalid Alsubhi, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.
†E-mail: kaalsubh@cs.uwaterloo.ca

proposed to abstract the information gathered from sets of alerts. However, these high-level techniques suffer from including alerts that are not significant, which often leads to inappropriate results. Therefore, low-level evaluation techniques are needed to automatically (or semi-automatically) examine large numbers of alerts and prioritize them, only allowing important alerts to receive further inspection. Accordingly, the reduced set of alerts leads to more precise, easier, and less resource-intensive high-level alert analysis.

It is not sufficient to propose new IDS alert management systems without studying their degree of efficiency in the presence of different configuration sets. The tuning of an IDS alert management system in order to ensure optimal performance remains amajor challenge. This hurdle is further complicated by the large spectrum of attacks that modern computer systems can be subject to. There is also a need to consider whether there exists a unique optimal configuration which works all the time or whether this optimal configuration changes depending on system state and administrative policies.

Building upon our previous work [1], this paper presents the specification and investigate configuration issues of FuzMet, a network intrusion detection system (NIDS) alert management framework which utilizes several metrics as well as a fuzzy-logic based approach for scoring and prioritizing alerts. The metrics are related to the applicability of the attack against the configuration of the network, the configuration status of the running IDSs, the importance of victim, and the relationship between the alert under evaluation and previous alerts. Additionally, FuzMet features a rescoring technique to dynamically rescore alerts based on the relationships between attacks. This leads to a further reduction of the number of alerts and provides a concise set of alerts to the system administrator for further investigation.

Of equal importance in the paper is the investigation of the impact of different configurations of the proposed metrics on the accuracy and completeness of the alert scores generated by FuzMet. Our approach is validated using the DARPA 2000 LLDOS 1.0 intrusion detection dataset. Comparative results between SNORT NIDS [2] alert scoring and FuzMet alert prioritization are presented. A considerable number of simulations were conducted in order to determine the optimal configuration of FuzMet, if any, with selected results presented and analyzed.

The paper is organized as follows. Section 2 discusses related work on alert management. Section 3 describes the FuzMet alert prioritization system. Section 4 presents the alert prioritization metrics used in FuzMet. Section 5 explains the fuzzy logic inference and its use in FuzMet. Section 6 explains the algorithm used for alert rescoring. Section 7 tackles the optimal configuration issue of the proposed metrics. Section 8 analyzes the complexity of FuzMet. Simulation results are presented and discussed in Section 9. We present the implicit attack graph generation from the prioritized alerts in Section 10. Finally, Section 11 concludes the paper.

The paper's contributions can be summarized as follows:

- FuzMet, an alert evaluation and prioritization framework that addresses limitations of previous works regarding alert ranking [3–6].
- Introduction of new metrics, such as sensor sensitivity, services stability, and alert relationship, which allows for better alert evaluation accuracy.
- A rescoring technique that dynamically scores alerts based on the relationship between attacks or the level of maliciousness of attackers.
- Application of fuzzy-logic reasoning to quantitatively score each alert based on metric values. To the best of our knowledge, we are the first to use fuzzy-logic for evaluating IDS alerts.
- A comprehensive study of the impact of different configurations of the proposed metrics on the accuracy and completeness of the alert scores generated by FuzMet.

2. RELATED WORK

It is common that an IDS generates a large number of alerts whenever suspicious activities are observed. A large number of alerts overwhelms the most expert of security administrators and makes it difficult to manually distinguish between realand false attacks. Two general approaches are used to deal with this problem. The first focuses on the monitoring device itself by enhancing its detection mechanism,

DOI: 10.1002/nem

eliminating the unnecessary rules, optimizing the signatures sets, and choosing the right location [7]. Although this solution promises to reduce the number of alerts, it requires prior in-depth knowledge by the security administrator of the detection mechanism. The second solution focuses on the sensor outputs. Several IDS alert management techniques fall into this category and include aggregation [8], clustering [9], correlation [10–12], and fusion [13]. Reducing the number of alerts by prioritizing critical attacks and discarding false alerts is the main objective of IDS alert management approaches. These techniques assist the security administrators in better assessing the alerts revealed by the IDS.

Techniques to construct attack scenarios fall into three classes. The first includes correlating alerts based on the similarity between their attributes, such as IP addresses and port numbers. Probabilistic alert correlation [11] falls into this category. The second class is based on the specification of a known sequence of attacks [14]. The third class is based on the exploration of dependencies between alerts by matching the consequences of a successful attack(s) with the prerequisites of a next-stage attack(s) [10]. The relationship metric we propose in Section 4.7 uses the similarity between alerts as well as the probabilistic alert correction to define an overall relationship of the alert with a predefined alert history window. This brings in the benefit of the ease of computation in addition to the detection of attacks based on either known [14] or unknown scenarios [10], without the overhead of maintaining a large knowledge base of attack graphs and/or prerequisite/consequence attack models. Our technique does not generate attack graphs as we are not interested in building them.

There have been some efforts in previous works for evaluating IDS alerts using some metrics [3–6]. However, these metrics have not been used all together, as proposed in this paper. Jinqiao Yu *et al*. [6] use an expert system to score alerts based on two criteria. First, the expert system checks whether an alert corresponds to a known attack in the vulnerability knowledge base. In the case of failure, the alert is considered as a new attack and gets prioritized for further investigation. In the other case, the alert corresponds to a known attack, in which case it receives a high score if it is applicable against the protected network, or a low score otherwise. Offering only two levels of alert scores is limited. FuzMet remedies this by allowing alert scores to belong to a continuous interval of values (from zero to ten). Similar to Jinqiao Yu *et al*., Qin and Lee [5] compute the alert score based on the severity of the attack and its relevance (applicability). Porras *et al*. propose the M-Correlator [4] alert ranking technique, in which alerts are ranked based on the likelihood of the attack to succeed (applicability), the importance of the targeted asset, and the amount of interest in the type of attack. Although these techniques are promising in the evaluation of alerts generated by signature-based IDSs, they cannot evaluate alerts raised by anomaly-based IDSs, since they rely heavily on the vulnerability knowledge base. FuzMet extends on these works by offering an alert scoring that works with both signature-based and anomaly-based IDSs and uses additional criteria, such as the sensor sensitivity, relationship between alerts, and service stability for more accurate evaluation of the alerts. Furthermore, the FuzMet approach differs from previous ones by providing a rescoring function of early non-critical attacks that prepare for later critical attacks. This way, the early steps of the attack will be prioritized for further analysis, such as correlation.

3. FuzMet ARCHITECTURE

As shown in the doted areas of Figure 1, the FuzMet alert management architecture involves three components: (a) data collection, (b) alert scoring metrics and inference, and (c) alert analysis.

3.1. Data collection

The data collection component includes the alert database, environment parameters, security administrator parameters and the vulnerability knowledge base. Alert attributes consist of several fields that provide information about the attack. This information varies from one IDS product to another. However, we assume that the alert structure is compatible with the Intrusion Detection Message Exchange Format (IDMEF) [15]. IDMEF is a standard data format for reporting alerts about suspicious events. Compared to other proposed formats for representing and exchanging IDS alert information, such as CIDF [16] and IDXP [17], IDMEF has the advantage of being XML based, and hence it provides flexibility for future extensions.

DOI: 10.1002/nem

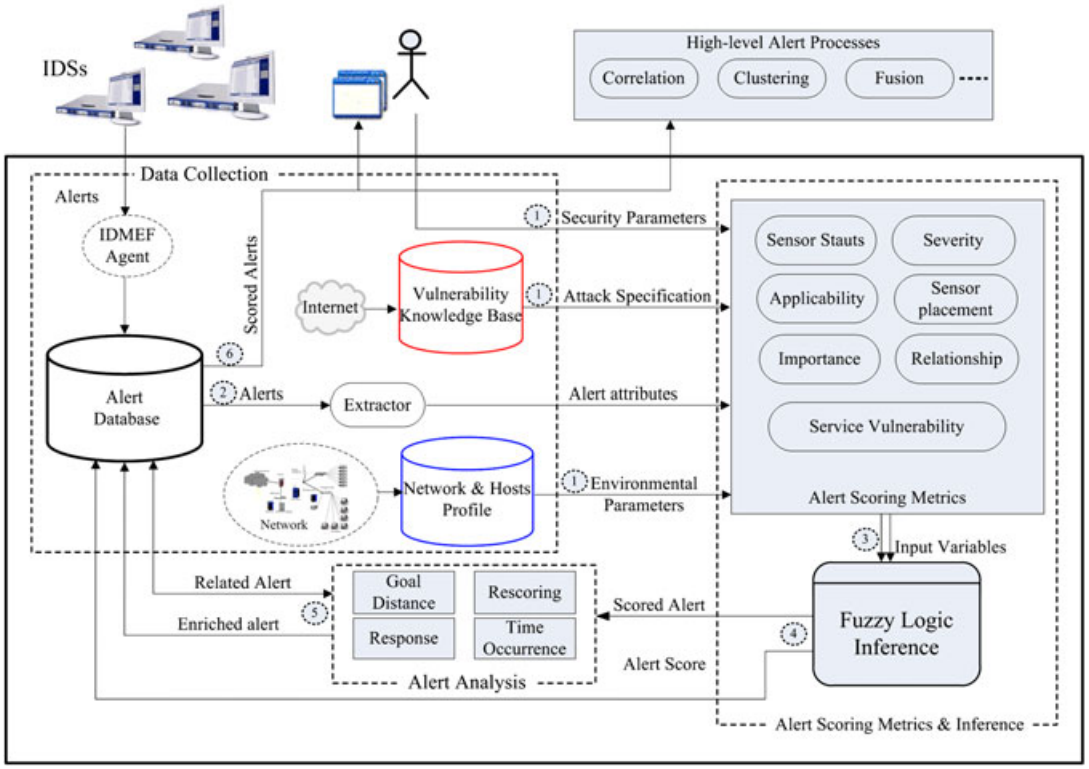

Figure 1. FuzMet alert scoring architecture

The security administrator can specify the environment configuration parameters to involve in the evaluation process. These can include, for instance, information about running services, applications, operating systems (plus versions), and existingvulnerabilities. Furthermore, the security administrator can specify further configuration parameters such as the importance of each host in the network, including the IDSs. Public vulnerability knowledge bases, such as the National Vulnerability Database (NVD) [18] and Bugtraq [19], contain detailed information about known attacks. The availability of such databases can help in the alert evaluation process. The data collection component makes the above resources available to the alert scoring metrics and inference component.

3.2. Alert scoring metrics and inference

The alert scoring metrics and inference component is a key element of the FuzMet architecture. Based on the information received from the data collection component, several metrics are computed and used as indicators to accurately evaluate the alerts. In this perspective, the computed metric values are passed to the fuzzy logic inference engine to calculate the overall alert score.

3.3. Alert analysis

The alert analysis component provides an additional evaluation of the IDS alert. This component hosts the functions of alert rescoring, measurement of distance to known attacks, occurrence time, and response plans.

As can be deduced from Figure 1, the alert evaluation process is carried out as follows. When a new alert is received, the associated scoring metrics are computed based on the available security parameters, the environmental parameters and the vulnerability knowledge base. Fuzzy logic inference is then employed in order to assign a score to the alert based on the metrics values. After that, the alert is stored in the alert database with its score. The alert is also passed to the alert analysis component for further investigation. The analysis component measures the distance of the current attack from its

DOI: 10.1002/nem

possible goals. It rescores the alerts that are suspicious to be a preparation step for later attacks. It also detects suspicious activities which violate predefined system usage (such as using a port number which is only allowed during working hours). Finally, it provides a response plan to the intercepted attacks and makes it available to the system administrator for further investigation.

4. ALERT PRIORITIZATION METRICS

The alert scoring metrics of Figure 1 are used to evaluate the criticality of alerts. Although some of these metrics have been individually used in previous works [3–6], they have not been used all together as proposed in this paper. Additionally, we define new metrics that help in accurately evaluating IDS alerts. The FuzMet scoring technique does not require all the metrics to be available during the evaluation. Intuitively, the presence of a large number of indicators will definitely increase the accuracy of the alert score. However, most of the metrics are easy to obtain, especially those that deal with protected environments and the vulnerability knowledge base. In the following, the alert scoring metrics presented in Figure 1 are detailed.

4.1. Applicability

The applicability process checks whether an attack that raised an alert is applicable to the current environment. Alert attributes are used by the vulnerability knowledge to determine target services. If at the time of the alert generation at least one of the target services is *running* on the machine with the destination IP/port of the alert, the target service is vulnerable to the attack, and there has been no installed patch to protect against it, then the alert is applicable.

In the case that the above conditions are satisfied except that the service is not running at the time of alert generation (e.g. machine off or service halted), a corresponding warning is generated to the security administrator.

4.2. Importance of victim

This metric indicates the criticality of the target machine reported in the alert. Several elements will participate in deciding the importance of the system in the environment including services, applications, and accounts. The goal of this metric is to increase the score of alerts related to suspicious activities that target critical system components, such as a main server. Before introducing the function that calculates the criticality of the target machine, we will present a general weighted equation that is used in this metric and the rest of the paper:

$$w'(a) = w(a) \times a \tag{1}$$

$$w(a) = \begin{cases} \text{low} & \text{if } 0 \le a < th_l \\ \text{med} & \text{if } th_l \le a < th_h \\ \text{high} & \text{if } th_{h \le a < 1} \end{cases} \tag{2}$$

where th_l and th_h represent a low and high threshold respectively. Equation (1) aims to compute the value of any element a based on its weight. A high weight is chosen if the object is critical and vice versa. In this metric, the criticality of a machine is calculated based on the running services/applications and the account associated with them. Different services have different weights according to their importance to the environment I(s) as well as the accounts I(Ac). Equation (3) gives the formula used for the importance metric:

$$I(m) = \frac{\sum_{s \text{ runs on } m} w'(I(s) \times I(Ac(s)))}{\sum_{s \text{ runs on } m} w(I(s) \times I(Ac(s)))} \tag{3}$$

DOI: 10.1002/nem

Importance describes the significance of the victim machine that is running in the protected environment. The value of the importance is calculated on a scale from zero to one. A zero indicates that the victim machine reported in the alert does not include any important host, service, application, account, or directory. Scores closer to one indicate that the attack is targeting a critical system component.

4.3. Sensor status

This metric measures the attack detection *efficiency* of each IDS in the network. The basic counters used for that are as follows:

- T_{c^+}: number of successful alerts (true positives)
- F_c^+: number of unsuccessful alerts (false positives)
- T_{c^-}: number of successful no alerts (true negatives)
- F_{c^-}: number of unsuccessful no alerts (false negatives)
- S_b: number of benign sessions
- S_m: number of malicious sessions

Let I denote the occurrence of an attack (intrusion) and $\neg I$ that of benign traffic. Similarly, let A ($\neg A$) denote the generation (absence) of an alert by the IDS. The efficiency of an IDS can be measured by two metrics:

1. $P(A/I)$, which is the conditional probability of generating an alert at the occurrence of an attack;
2. $P(A/\neg I)$, which represents the false detection rate; i.e. the probability that the IDS generates alerts in the absence of an attack.

We adapt the method proposed by Axelsson [20] to compute the probability $P(I/A)$ that the raised alert A was that of an actual intrusion. Taking into consideration that the probability of intrusion $P(I)$ is very low, as attack flows are only a tiny fraction of the total number of flows, we denote $P(I)$ by ϵ; then $P(I/A)$ can be expressed as follows:

$$P(I|A) = \frac{P(A|I)}{P\left(A|I) + \frac{1|-\epsilon}{\epsilon} P(A|\neg I)\right.} \tag{4}$$

This equation shows that the impact of the rate of false positives $P(A|\neq I)$ is important and the system has a good predictability of correct alert generation only if $P(A|\neg I)$ is small enough to cancel the effect of $1/\varepsilon$.

From the alert prioritization perspective, $P(A/I)$ is the rate of correct alerts that scored high over the total number of observed attacks. This can be calculated as the ratio of correct alerts to the total number of attacks, i.e.

$$P(A|I) = \frac{T_c^+}{S_m} \tag{5}$$

The false detection rate $P(A/\neg I)$ is computed as follows:

$$P(A|\neg I) = \frac{F_c^+}{S_b} \tag{6}$$

The *efficiency* of the sensor is affected by a number of parameters including the sensor configuration, rule set version, software release version, and the set of installed patches.

4.4. Sensor placement

The importance of the placement of an IDS in a network is related to the importance of the machines and services running in the sub-network it protects. For example, an IDS protecting a demilitarized zone (DMZ) containing the companyweb and mail servers has a critical (high) placement compared to another

DOI: 10.1002/nem

IDS protecting a cluster of low-grade staff machines. The process of alert prioritization will give precedence to high score alerts coming from the first IDS over high scorealerts registered by the latter.

4.5. Severity

The severity score metric SS*(a)* measures the risk level posed by a particular vulnerability a. There are several knowledge base sources which provide severity scores for known attacks, including MITRE CVE, NIST, Secunia, and OSVDB, as well as software-developer specific severity score databases. The severity score value may vary from one organization to another. For instance, the FileZilla unspecified format string vulnerability has been reported in NIST as a very severe vulnerability scored 10 out of 10, while Secunia reported this vulnerability to be moderately critical. In order to emphasize an attack when at least one source classifies it as *high*, the weight functions w and w' are used equations (1) and (2). A confidence factor $\delta(i, a)$ is used to reflect the measure of trust given to a severity score $\mathrm{SS}_i(a)$ based on the source i of the score as well as the victim machine/services associated with alert a. For example, if attack a targets a Microsoft SQL server, then the severity score of a which is provided by Microsoft is given a higher trust than the severity scores provided by other sources. Based on this, the formula for the severity score of an attack a is given by

$$\mathrm{SS}(a) = \frac{\sum_{i=1}^{n} w'(\mathrm{SS}_i(a)) \times \delta(i,a)}{\sum_{i=1}^{n} w(\mathrm{SS}_i(a))} \tag{7}$$

4.6. Service vulnerability

We adapt the method proposed by Abedin *et al*. [21] to analyze only the service that the attacker is targeting. This method is used to calculate a unified score representing the strength or weakness of the targeted service. The result is then used in the overall alert scoring.

It is possible to measure the vulnerability score *V(s)* of a service s which appears in alert a based on the current vulnerability score $V_e(s)$, the past vulnerability score $V_h(s)$, and the release time *T(s)* of the service (e.g. number of months).

A raised alert a explicitly mentions the targeted service s (e.g. OS or service/application) that the attacker is trying to violate. In most cases, the targeted service can be determined from the information on port numbers.The set of all vulnerabilities of a service can be divided into historical vulnerabilities $V_h(s)$, which are past flaws that have been removed either through patching or new software releases, or existing vulnerabilities, which are the currently known security flaws to which no solution has yet been provided or the corresponding patch has not yet been applied.

V_e is computed as follows:

$$V_e(s) = \frac{\sum_{v \in V_e(s)} w'(\mathrm{SS}(v))}{\sum_{v \in V_e(s)} w(\mathrm{SS}(v))} \tag{8}$$

where $\mathrm{SS}(v)$ is the severity score of vulnerability v.

For the historical vulnerability score $V_h(s)$ of service s, vulnerability knowledge bases, such as MITRE CVE, are consulted to measure how stable the service was in the past. We use a decaying factor to give less weight to older vulnerabilities. Equation 9 shows how $V_h(s)$ is computed:

$$V_h(s) = \frac{\sum_{v \in V_h(s)} w'(\mathrm{SS}(v)) \times \lambda^{-\mathrm{age}(v)}}{\sum_{v \in V_h(s)} w(\mathrm{SS}(v))} \tag{9}$$

Finally, the overall vulnerability score *V(s)* of a service s is computed as follows:

$$V(s) = \frac{T(s) \times V_e(s) + V_h(s)}{1 + T(s)} \tag{10}$$

DOI: 10.1002/nem

This formula ensures that the older the last release time of the service, the more weight is given to the score of existing vulnerabilities. Conversely, the more recent it is, the more confidence is given to historical vulnerabilities.

4.7. Alert relationship

Usually attackers use multiple steps in order to achieve their final goal. Computing the final score of an alert therefore needs to involve the relationship it has with previous ones. We increase the score of an alert if we find that it has a strong relationship with other stored alerts. In order to avoid lengthy computations, we restrict the search to those alerts that occurred within a defined history window RW. This restriction helps in discarding very old alerts and focusing on recent ones. We assume this approach is valid based on the observation that attackers typically try to achieve their goal as soon as possible before they can be identified.

The relationship between two alerts a_i and a_j is based on computing the similarity between their respective source IP addresses ($\mathrm{Sim}_{\mathrm{sip}}$), destination IP addresses ($\mathrm{Sim}_{\mathrm{dip}}$), source ports ($\mathrm{Sim}_{\mathrm{spt}}$), and destination ports ($\mathrm{Sim}_{\mathrm{dpt}}$). The source IP addresses similarity $\mathrm{Sim}_{\mathrm{sip}}$ is computed based on equation 11. A similar formula exists for $\mathrm{Sim}_{\mathrm{dip}}$. $\mathrm{Sim}_{\mathrm{spt}}$ and $\mathrm{Sim}_{\mathrm{dpt}}$ produce one if the port numbers match and zero otherwise:

$$\mathrm{Sim}_{\mathrm{sip}}(a_1, a_2) = \frac{\sum \text{similar bits}(S_{\mathrm{IP}}(a_1), S_{\mathrm{IP}}(a_2))}{\sum \text{all bits}(S_{\mathrm{IP}}(a_1), S_{\mathrm{IP}}(a_2))} \tag{11}$$

The overall similarity is a weighted sum of the four similarities mentioned above, with the highest weight given to the source IP address similarity and the lowest to the source port number.

The relationship between two alerts equation (12) is then computed by taking into account the *follow-up* probability, which is a measure of the probability that an attack of type $T(a_j)$ is followed by an attack of type $T(a_i)$. The values of these are taken from the statistical analysis made by Valdes *et al*. [11]:

$$R(a_i, a_j) = P(T(a_j), T(a_i)) \times \mathrm{Sim}(a_i, a_j) \tag{12}$$

Finally, the relationship score of an alert a_i (equation 13) is equal to the maximum relationship score it has with the alerts that occurred at most RW time units before a_i (i.e. $0 < ts(a_i) - ts(a_j) < \mathrm{RW}$):

$$R(a) = \max_{0 < ts(a_i) - ts(a_j) < \mathrm{RW}} R(a_i, a_j) \tag{13}$$

5. FUZZY LOGIC INFERENCE

A fuzzy logic system reasons about the data by using a collection of fuzzy membership functions and rules. It makes clear conclusions possible to derive from imprecise information. In this regard, it resembles human decision making because of its ability to work with approximate data and find precise results. Fuzzy logic differs from classical logic in that it does not require a deep understanding of the system, exact equations, or precise numeric values. It incorporates an alternative way of thinking, which allows for complex modeling of systems using a high level of abstraction of gained knowledge and experience. Fuzzy logic allows the expression of qualitative knowledge, including phrases such as ‘too hot’ and ‘not bad’, which aremapped to exact numeric ranges [22,23].

There are a number of steps that the fuzzy logic inference is composed of. Figure 2 illustrates these steps using a numerical example. The steps can be summarized as follow. First, the fuzzy logic inference system receives input values that represents themeasurement of the parameters to be analyzed and evaluated. Then, these values are subjected to fuzzy rules in order to obtain output values for each individual rule. These outputs will then be weighted and averaged in order to have one single output value. Finally, the averaged value will be defuzzified in order to map the fuzzy decision value into a crisp value.

Because of its ability to work with approximate data, we use a fuzzy logic engine to reason about IDS alerts. As illustrated in Figure 2, results coming from the metrics presented in the previous section

DOI: 10.1002/nem

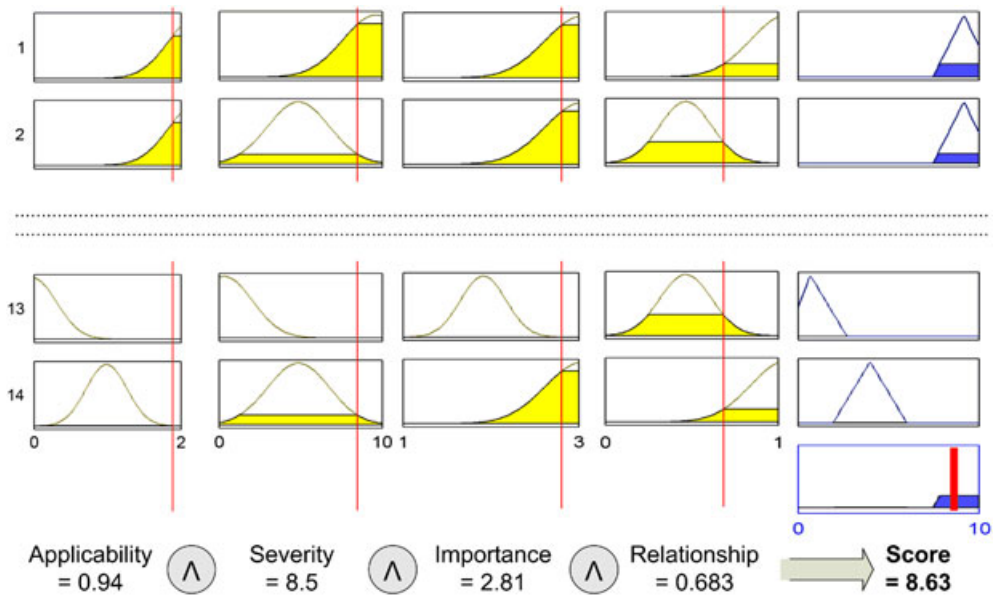

Figure 2. Example of fuzzy logic inference in FuzMet

are used as input to the fuzzy logic inference engine in order to investigate the seriousness of the IDS alerts and assign a single score for each alert.

To develop the fuzzy logic inference, we need to define two main components. The first one is the definition of the membership functions of all input and output parameters. The second component is the design of the fuzzy rules which formulate the conditional statements of the input values of the parameters and determine their affect on the outputs.

5.1. Membership functions

The membership function represents the contribution level of each input parameter graphically. It associates a weighting with each of the inputs, defines functional overlap between inputs, and determines an output response. The membership function is often represented by a triangular, Gaussian, trapezoidal, sigmoidal, or polynomial function. We use a Gaussian membership function to graphically describe the three fuzzy sets (i.e. Low, Med, and High) for each input parameter. The Gaussian membership function is chosen for specifying fuzzy sets because of its smoothness and concise notation in designing linguistic variables [24]. For each output parameter, four triangular membership functions were designed (i.e. VeryHigh, High, Med, and Low) and distributed in the range [1.0, 10.0].

5.2. Fuzzy rules

Fuzzy rules determine the influence of the input membership values on the output sets. Rules in a fuzzy expert system are usually extracted based on leveraging the domain expert knowledge, which can be in the following form:

if *applicability* is *High* and *severity* is *Average*
then set *alert score* to *High*

where *applicability* and *severity* are the input variables, *score* is the output variable, *High* is a membership function (fuzzy subset) defined on *applicability*, *Average* is a membership function defined on *severity*, and *High* is a membership function defined on the alert *score*. Table 1 gives the set of fuzzy rules which we used in the experiments.

DOI: 10.1002/nem

Table 1. Fuzzy rules set

Rule	Applicability	Severity	Importance	Relationship	Output
1	High	High	High	High	V. high
2	High	Med	High	Med	V. high
3	Low	Low	Low	Low	Low
4	Low	Low	Med	Med	Low
5	Med	Med	High	High	Med
6	Med	Med	High	High	Med
7	Low	High	Low	Low	Low
8	High	Low	Low	Low	Low
9	Low	Med	Low	Low	Low
10	Low	Med	Low	Med	Low
11	Low	Med	Low	High	Low
12	Low	Low	Low	High	Med
13	Low	None	None	None	Low
14	High	Med	High	Med	Low

In FuzMet, fuzzy logic is used to score alerts generated by the different IDSs. As shown in Figure 2, the fuzzy logic inference engine first takes the input values from the different metrics (e.g. applicability, severity, importance, and relationship metrics), then *fuzzifies* them using the membership functions. The fuzzy rules are then triggered to generate the output sets. The outputs are then averaged into a single fuzzy value which will be *defuzzified* in order to give a crisp value that represents the seriousness of the alert.

6. ALERT RESCORING

Alert management techniques such as aggregation, grouping, scoring, filtering, clustering, correlation, and fusion were proposed to deal with and abstract large numbers of alerts. First, alerts are aggregated from multiple IDSs, then similaralerts are grouped together into meta-alerts [12]. The scoring function evaluates the meta-alerts and assigns a score to each one according to its importance. Low-scored alerts are usually discarded and do not get involved in any further analysis. Correlation functions may then be applied to present the attack scenarios.

The goal of the alert rescoring algorithm we propose is to rescore alerts that have already been scored based on their relationship with a given new alert. One of the reasons for rescoring alerts is to notify security administrators of the early steps of the attack, which may have received low scores. This will emphasize preliminary activities of an attacker which may help in predicting the forthcoming steps of the attack. Scoring alerts only once limits the identification of the non-critical early steps of an attack, especially if a threshold mechanism is employed to filter out low-scored alerts. Also, high-level alert management techniques, such as correlation and clustering, can benefit from the rescoring and thus provide more accurate results.

As an example, consider the case of an attacker who first scans a victim machine using an *IPSweep*. This probing activity is likely to receive a low score. Later, if the attacker launches a *SadmindBuffer-Overflow* attack based on the vulnerabilities discovered in one of the previously scanned machines, this new attack is likely to receive a high score because of its seriousness. The security administrator will not be able to see the early steps of the attack if a filtering operation is applied. On one hand, involving only critical alerts in the high-level operations, such as correlation, prevents the non-critical early steps of the attack from being considered. On the other hand, involving all alerts in the high-level operation leads to an overwhelming number of correlated alerts that are difficult to manage. Hence it is important to highlight only the critical alerts and those related to them.

The rescoring of alerts which we propose is based on the *prepare-for* relationship [11]. The *prepare-for* relationship checks if there is any relationship between the currently evaluated alert and the previous alerts in the alerts log. Security analysts can apply the rescoring function periodically (e.g. every night or weekly) or just prior to applying the high-level management techniques.

DOI: 10.1002/nem

The alert rescoring algorithm takes as input the alerts log, the score threshold for prioritizing alerts th_s, and the relationship time window interval W_r. The algorithm scans from the last alert down to the first one or to some predefined rescoring depth R_{depth}. For each high-scored alert a_i in the log, we calculate its relationship Section 4.7 with each low-scored and applicable Section 4.1 alert a_j which preceded a_i with at most W_r time units. If the relationship is strong then a_j is rescored to a higher value according to the strength of the relationship, based on the following formula:

$$\text{Score}(a_j) = \max_{0<ts(a_i)-ts(a_j)<\text{RW}} \left(th_s, R(a_i, a_j) \times \text{Score}(a_j)\right) \tag{14}$$

7. FuzMet configuration issues

FuzMet uses a number of metrics, fuzzy inference rules, and alert rescoring mechanisms, all of which have several configurable parameters that influence the precision of intrusion detection and alert prioritization. The security administrator has a number of parameters that can be configured, while others are not. Non-configurable parameters include the severity score values $\text{SS}_i(a)$ gathered from security expert organizations, the sensor update status and accuracy, and the attack follow-up probability matrix P equation (12). The set of configurable parameters includes the five parameters of the weight function w (equation 2), which in turn influence the machine importance metric *I(m)* equation (3), the severity score metric SS equation (7), the existing vulnerability metric VS_e equation (8), and the historical vulnerability metric VS_h equation (9). The importance *I(s)* of each service s and the importance of each user account *I(a)* needs also to be configured in order to reflect the criticality of each service, user account, and machine. The sensor status metric has four configurable weight parameters. The severity score metric equation (7) requires the definition of $\delta(i, v(a))$ for each severity score provider i and targeted victim *v(a)* tuple. This parameter can be made into a more static form δ_i if only a single trust value is given to a severity score provider independent of the target victim. The service vulnerability metric equation (10) has an additional four parameters including the decay coefficient λ as well as the three η_i weights. The relationship metric equation (13) has also four additional parameters. Adding to the parameters relatedto the rescoring process, the overall number of configurable parameters of FuzMet becomes

$$N_{\text{cfgp}} = 6 + |services| + |accounts| + |scoresources| \times |victims| \tag{15}$$

Besides metric configuration, there is a need to define the appropriate fuzzy inference rules which help in capturing the severity of each attack. As can be noticed, providing an optimal configuration of the FuzMet system is not trivial, to say the least.

8. COMPLEXITY ANALYSIS

In this section we investigate the complexity generated from using FuzMet alert prioritization as it is important for an alert prioritization mechanism to have a light overhead. Studying the complexity analysis for calculating the values of each metrics as well as for the fuzzy-logic reasoning is important to guarantee the real-time alert evaluation capability. However, real-time alert evaluation is not essential in IDSs, unlike IPSs, which require real-time alert evaluation and a prioritization mechanism for immediate response.

Let N be the current number of generated alerts and M the number of machines in the protected network. For the DARPA 2000 dataset, N is equal to 3502 and M to 23.

The *applicability* metric Section 4.1 checks whether an attack is applicable or not to the target machine/service. This requires one access to the services database using the attack destination IP/port to get the target service, one accessto the vulnerability knowledge base, and possibly a third access to check whether there are patches against the attack in case the service is recorded as vulnerable in the vulnerability knowledge base. Hence the applicability metric induces $2sN$ to $3 \times N$ database accesses.

This number can be further reduced to $1 \times N$ if the system maintains a local vulnerability knowledge base, which contains a table of all <IP, port, service, alert type, applicable> tuples. If a service s is

DOI: 10.1002/nem

patched against a given vulnerability v then the entry will have 0 for the applicability, and 1 otherwise. The local vulnerability knowledge base can be created beforehand (negligible overhead) or as new alert types are detected. It then needs to be updated each time a change occurs in the list/configuration of installed services/patches.

The *importance of victim* metric equation (3) computes the criticality of a given machine in the protected network. Once computed, the importance of some machine m can be cached (or stored back to the system structure database). Hence there is no need to compute a second time except for the low-frequency case where the set of active services/accounts for m changes, in which case $I(m)$ is invalidated and needs to be recomputed. Overall the complexity induced by I is $O(N+M)$, where M is the number of machines in the protected network.

The *severity score* metric equation (7) requires, for each alert, the retrieval for each severity score provider i, of the confidence factor $\delta(i,a)$ and the severity score $\mathrm{SS}_i(a)$. If I is the number of information sources (assumed small), then the overhead induced by the severity score metric is $3I \times N = O(N)$. The actual overhead can be further reduced if caching is used to exploit the redundancy of alert types. For example, for the DARPA2000 dataset this number is equal to 14 compared to a total number of alerts of 3502; thus a ratio of $1/265 = 0.4\%$. Note also that the overhead due to multiplication is $(2IN)$, summation $(2(I-1)N)$, and division (N) is negligible compared to database access operations.

The service *vulnerability score* (equation 10) is only related to the set of currently running services in the system, which is independent of the number of generated alerts. Hence it it does not incur an overhead at runtime as it can be computed offline.

The alert *relationship metric* equation (13) checks for each alert the set of all alerts that preceded it within the RW interval. This can induce a considerable overhead when a burst of alerts is generated. This explains the peaks in the alert processing times Figure 3 in the conducted experiments. The peaks in metric computation are, however, still small (maximum peak in the experiment is 16 0ms). In practice, we limit the number of alerts to be checked by the relationship metric to a maximum value to avoid the case when an overwhelming number of alerts is generated in a small time interval or in the case that a wrong configuration sets RW to a very large value.

The *rescoring process* suffers from the same peak problem as the relationship metric. However, it is less of an issue because this process is conducted offline.

From the previous analysis, it can be noted that FuzMet has an overall small complexity and hence can be used without concern as a runtime alert prioritization engine.

9. EXPERIMENTAL RESULTS

In order to validate the effectiveness of the FuzMet approach for alert prioritization, we conducted a number of experiments. We installed SONORT to scan the DARPA 2000 LLDOS dataset 1.0 [25] and stored the generated alerts in a MySQL database. Java was used to compute the different metrics related to the alerts. These metrics were then input to the fuzzy rule set of Table 1 in order to generate FuzMet alert scores. The fuzzy rules used in our experiment are written based on the information provided regarding the DARPA dataset. The goal of designing these fuzzy rules is that they should prioritize the critical alerts among a large number of alerts mixed with a high rate of false positives. Because the used dataset does not come with full information regarding the protected environment, we design the fuzzy rule set based on the knowledge of detecting the documented attacks and the relationship of the metrics used to detect these attacks. Matlab Fuzzy Logic toolbox [26] wasused for the fuzzy rules specification and experiments. We compared our results to the alert scores generated by SNORT.

In fact, the lack of other appropriate datasets that satisfies the common research requirements makes the DARPA dataset one of the major contributions in evaluating IDSs. The datasets have been used extensively by researchers to evaluate various types of IDS algorithms. The DARPA dataset has been criticized for the way it was designed and its obsolescence and incapability of including new sorts of network attacks. Despite the criticisms, we argue that it is still useful for evaluating IDS alert management techniques, such as alert correlation, clustering, and merging. The use of the DARPA dataset in our work was not for detection purposes but to evaluate the output of the detection approach (SNORT

DOI: 10.1002/nem

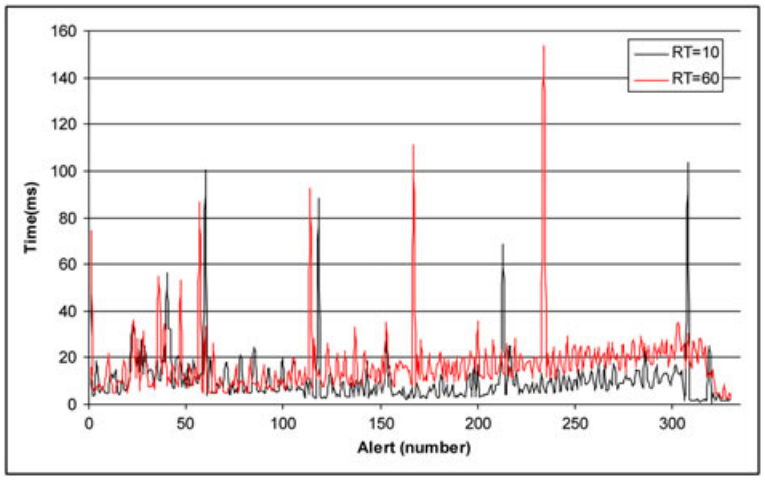

Figure 3. Alert processing time

in our case) for the sake of assisting the security administrator in quickly and easily discovering important and relevant alerts.

The DARPA LLDOS 1.0 dataset contains traffic collected from two network zones: 'DMZ' and 'inside'. The series of attacks in the dataset are carried over multiple sessions or phases, the interval times of which are shown in Table 2. The first phase scans the network in order to launch a distributed denial-of-service (DDoS) attack against an off-site server. The second phase looks for the *Sadmind* daemon of live IPs. The third phase exploits a Sadmind vulnerability. The fourth phase proceeds by

Table 2. Prioritized alerts of the DARPA 2000 LLDOS 1.0 dataset using FuzMet

					Non-grouped			Grouped		
Phase	From	To	Length	Alert name	# Alerts	# High scores	Avg	# Alerts	# High scores	Avg.
Phase 1	9:51:36	9:52:00	0:00:24	ICMP Echo Request	786	0	≈2	29	0	≈2
				ICMP Echo Reply	30	0	≈1.5	12	0	≈1.5
Phase 2	10:08:07	10:17:10	0:09:03	RPC portmap sadmind request UDP	250	79	≈8	46	28	≈8
				RPC sadmind UDP PING	9	6	≈5	4	3	≈5
Phase 3	10:33:10	10:35:01	0:01:51	RPC sadmind with root attempt UDP	46	28	≈9	40	29	≈9
				RPC sadmind UDP NETMGT_PROC _SERVICE	46	6	≈9	3	3	≈9
Phase 5	11:26:15	11:41:19	0:15:04	BAD-TRAFFIC loopback traffic	141	141	≈9	1	1	≈9
False alerts				NETBIOS NT NULL session	2	0	≈1	1	0	≈1
				ATTACK_RESPONSES Invalid URL	4	0	≈1	1	0	≈1
				SNMP request udp	12	0	≈6	9	0	≈6
				SNMP public access udp	6	0	≈5	6	0	≈5
				ATTACK-RESPONSES 403 Forbidden	10	0	≈1	3	0	≈1
				MS-SQL version overflow attempt	1	0	≈1	1	0	≈1
				ICMP redirect host	2159	0	≈1	26	0	≈1
Total					3502	260		182	64	

DOI: 10.1002/nem

installing an mstream Trojan on the compromised machine. Finally, the last phase launches the DDoS attack against the remote site.

This section is divided into four parts. The first part shows the output of SNORT and FuzMet for one configuration set and details the comparison between them. The second shows the results of the rescoring technique. The third part studies the impact of FuzMet on the accuracy of the IDS. Finally, the fourth one focuses on the optimal configuration problem and presents the result of a selected set of results from among a set of 200 conducted simulations.

9.1. Alert scoring results

SNORT was used with the maximum detection capability to scan and detect intrusions within the binary tcpdump file of both of the 'inside' and 'DMZ' traffic. SNORT reported 3502 alerts (321 inside, and 3181 DMZ).

In order to filter out redundant alerts, we employed a grouping of alerts based on exact similarity within a specific window of time. This resulted in a new total of 156 alerts Table 2 and thus a gain of 95.5%.

The FuzMet scoring technique was applied to the alerts generated by SNORT. For each alert, we compute the value of all the metrics we defined earlier, except for the sensor status and service vulnerability metrics, because the used dataset does not provide knowledge about the status of the targeted services and applications running over the evaluation network. However, the other metrics were good enough to prioritize the most critical alerts. The attacker in the first phase tries to scan the networkby employing the Internet Control Message Protocol (ICMP) echo request, looking for 'up' hosts. SNORT generates 816 alerts as a response to attacker's ICMP requests and the hosts ICMP replies. FuzMet evaluated these incidents and scored them aslow (1.2–2.3), as shown in Table 2. In the second phase, we received 259 alerts from the traffic of both the *DMZ* and *inside* parts, which represents the attacker's attempts to probe the discovered live hosts from the previous phase to determine which hosts are running the *Sadmind* remote administration tool. We scored these alerts differently based on the context in which they occurred. For instance, the 'RPC portmap Sadmind request UDP' alert that was triggered by the activity targeting the inside firewall interface is scored low. However, this alert is scored high when the target host is running a *Sadmind* service. The remote-to-root exploit has been tried several times in the third phase and SNORT raised 92 alerts, of which FuzMet prioritized 34. Since we focus on evaluating alerts generated by network IDSs, we did not involve the audit data from the hosts in the network, and therefore phase 4 was not included. The DDoS attacks in phase 5 triggered 141 alerts which FuzMet prioritized as critical events.

Table 2 summarizes the results of the FuzMet alert scoring (with and without the grouping function) technique on the DARPA dataset. FuzMet alert prioritization was effective in identifying the false positive alerts which SNORT failed to detect. For example, SNORT generates an 'MS-SQL version overflow attempt' alert with the highest priority, but we scored this alert low based on our criteria since the target address is running a Mac operating system and this attack is impossible to succeed in this context. Figure 4 shows that after we score the alert a security administrator can be provided with the most important alerts, unlike the result of SNORT, which assigns a level-two priority (out of three) to most of the alerts.

9.2. Alert rescoring results

We applied FuzMet rescoring to the stored alerts that were previously scored. As we discussed earlier, in Section 6, a simple alert grouping technique was applied to the alert log to remove redundancy (similar alerts which are close in time). This technique groups together the alerts that are similar in their IP addresses, port numbers, and attack type. The grouped alerts represent all the attacker steps used to launch the DDoS attack. Since the LLDOS 1.0 dataset consists of only one complete attack scenario, the first phase, which contains non-critical attacks (regular scanning), is a good candidate for rescoring, for two reasons: first, because it is a preparation step for later attacks; second, FuzMet

DOI: 10.1002/nem

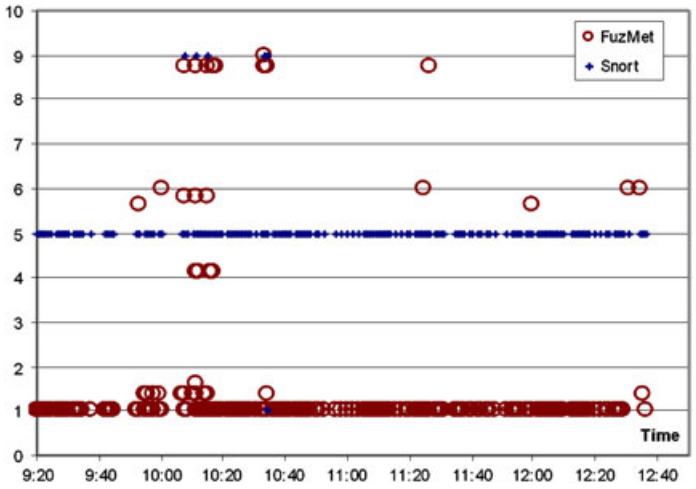

Figure 4. FuzzMet versus SNORT scores. SNORT scores most alerts as medium, whereas FuzMet manages to filter out unnecessary alerts, emphasize the important ones, and implicitly identify the different phases of the compound attack

and SNORT score the alerts generated by this phase as low. Therefore we focused our analysis on the alerts related to the first phase to check the usefulness of our rescoring approach.

In phase 1, the attacker starts to scan the network at 09:51:36 until 09:52:02 by performing a scripted IPsweep of multiple class C subnets on the victim network. Previously, this phase was scored low by FuzMet (as well as SNORT) according to its seriousness and impact. FuzMet successfully rescored this phase to be attached with the other phases of the DDoS attack. As shown in Figure 5, the alerts related to phase 1 were scored between 1.35 and 4.38 but after applying the rescoring function their scores increased to 8. The rescoring values was chosen to be 8 to ensure that the rescored alerts would be included in the prioritized alert set (assuming that the score threshold for prioritizing alerts *ST* is 7). As a result, all thecritical alerts as well as the preparation steps have been prioritized and presented to the security analyst.

9.3. Impact on sensor accuracy

The studied dataset contains 70 228 sessions (34 890 inside and 35 338 DMZ), of which there were 33 885 malicious sessions for the inside part and 34 833 malicious sessions for the DMZ part.

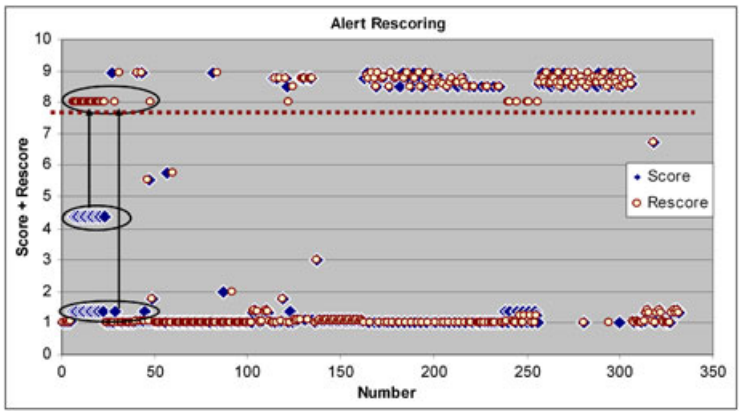

Figure 5. Alert rescoring results. FuzMet rescoring manages to identify the first phase of the compound attack, which both SNORT and FuzMet initial scoring missed.

DOI: 10.1002/nem

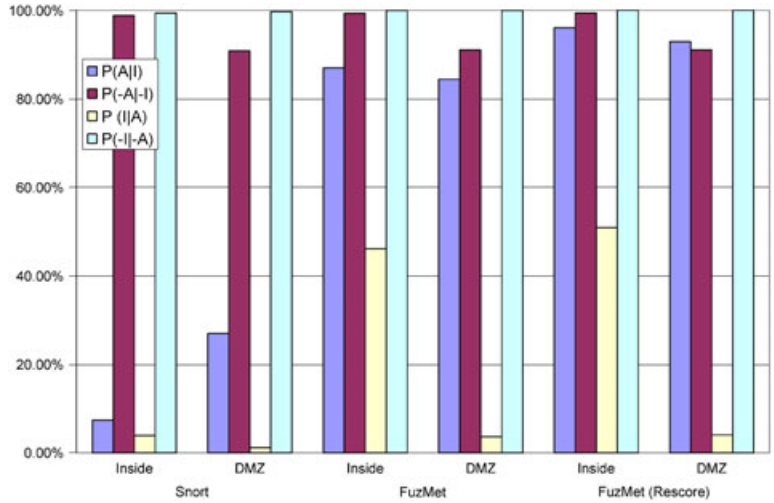

Figure 6. Enhancement of detection accuracy using FuzMet and rescoring

As can be seen from Figure 6, the rate of true positives $P(A|I)$ for the inside traffic changes from 7.39% using SNORT to 86.96% after using FuzMet scoring, and goes up to 96.09% after applying the rescoring algorithm. This implies the considerable enhancement of 1200%. The enhancement rate for the the DMZ traffic part reaches 245% for FuzMet with rescoring.

The rate of true negatives $P(\neg A|\neg I)$ also improves for the inside part by 0.53% using FuzMet and 0.6% with rescoring. For the DMZ traffic, the rate of true negatives makes the slight improvement of 0.25% and 0.29% with rescoring.

The above results are quite important and prove that using FuzMet with rescoring can lead to considerable enhancement in the accuracy of a sensor.

It may appear, based on the studied dataset, that the difference of enhancement between the use of FuzMet and that of adding the rescoring mechanism is not much. However, the rescoring always provides an additional enhancement which may be considerable for other datasets.

9.4. FuzMet configuration issue

As discussed in Section 7, the parametrization of the FuzMet alert scoring system can be critical to its effectiveness. Misconfiguration leads to imprecise outcomes, which consequently results in missed attacks. The objective is therefore todetermine a good configuration of the FuzMet set of parameters.

Because the used dataset does not come with information about network topology, the set of running services, and the different user accounts, a number of FuzMet metrics could not be involved in the experiments. These include the sensor status, sensor placement, and service vulnerability metrics. The metrics which were used are the applicability, importance, severity, and relationship metrics. For simplicity, all severity score sources were given equal confidence $(\delta(i, v(a)) = 1)$ equation (7).

Two hundred simulations have been conducted with different configuration parameters in order to determine the best configuration set. A good configuration set is one which makes FuzMet prioritize all and only those alerts which actually belong to theDDoS attack phases. Owing to space limitations, and because it is not possible to aggregate simulation results, only four representative simulations will be shown in this section. The goal in choosing these four simulations is to show the impact ofthe parameter configuration on the effectiveness of FuzMet. Intuitively, the parametrization of the FuzMet alert scoring system determines its effectiveness. Misconfiguration can easily lead to imprecise outcomes, which consequently result in missed attacks. These four simulations demonstrate how misconfiguring FuzMet can severely affect its performance.

DOI: 10.1002/nem

Table 3 shows the parameters of the severity and relationship metrics for the selected simulations. The weight group contains the configuration parameters for the weight function which directly affects the severity score metric equation (7). Table 4 shows the configuration parameters related to the importance metric. A value of 1 indicates lower importance, while a value of 3 indicates the highest importance. A zero value indicates the case where the target machine is unknown or that the value has not yet been configured.

Simulation S_1 generated the worst result among the 200 simulations, with only one single alert given high priority Figure 7a. The result can be explained by the fact that the machines running the Sadmind service, core to the DDoS attack, have been assigned a null importance (172.16.112.10, 172.16.112.50, and 172.16.115.20). In S_2 (Figure 7(b)), 31 alerts have been assigned high priority. S_2 differs from S_1 mainly in the importance metrics where the machines running Sadmind have been assigned non-zero importance in S_2.

S_3 gives a high priority to 11 alerts only (Figure 7(c)). However, if the high-priority threshold is lowered from 8 down to 7, S_3 records a number of 66 high-priority alerts, hence equaling those generated by S_4 (Figure 7(d)). S_4manages to generate the best result among the 200 simulations with a number of 66 high-score alerts and the highest average score of 4.02. S_3 and S_4 have the same importance configuration and differ in the weight and relationship parameters.

Table 3. Metric parameters

Parameter		S1	S2	S3	S4
Weight	low	0.10	0.34	0.35	0.16
	med	0.49	0.53	0.40	0.45
	high	0.83	0.56	0.45	0.60
	thl	0.30	0.30	0.30	0.30
	thh	0.60	0.60	0.60	0.60
Relationship	sip	0.92	0.73	0.25	0.84
	dip	0.65	0.59	0.25	0.63
	spt	0.10	0.29	0.10	0.19
	dpt	0.22	0.29	0.10	0.11
	RW	10	10	5	20

Table 4. Importance configurations

Dest. IP	Target	S1	S2	S3	S4
131.84.1.31	Unknown interface	2	2	1	1
172.16.112.1	Unknown interface	3	1	2	2
172.16.112.10	Sun Solaris 2.6	0	1	3	3
172.16.112.100	Windows NT 4.0	3	0	2	2
172.16.112.105	unknown	1	3	1	1
172.16.112.194	Sun Solaris 2.5.1	1	1	2	2
172.16.112.50	Sun Solaris 2.5.1	0	3	3	3
172.16.113.148	Linux Redhat 5.0	1	3	2	2
172.16.113.204	Sun Solaris 2.5.1	0	0	2	2
172.16.113.84	SunOS 4.1.4	0	1	2	2
172.16.114.10	Sun Solaris 2.6	3	1	3	3
172.16.114.2	Sidewinder	0	1	1	1
172.16.114.20	Sun Solaris 2.7	1	2	3	3
172.16.114.30	Sun Solaris 2.7	2	0	1	1
172.16.114.50	Linux Redhat 4.2	1	0	2	2
172.16.115.1	Firewall interface	2	1	1	1
172.16.115.20	Sun Solaris 2.7	0	2	3	3
172.16.115.87	Windows 95	0	1	2	2
172.16.116.44	Windows 3.1	3	3	2	2
172.16.117.103	MacOS	2	3	2	2
172.16.117.111	Mac OS	1	1	2	2

DOI: 10.1002/nem

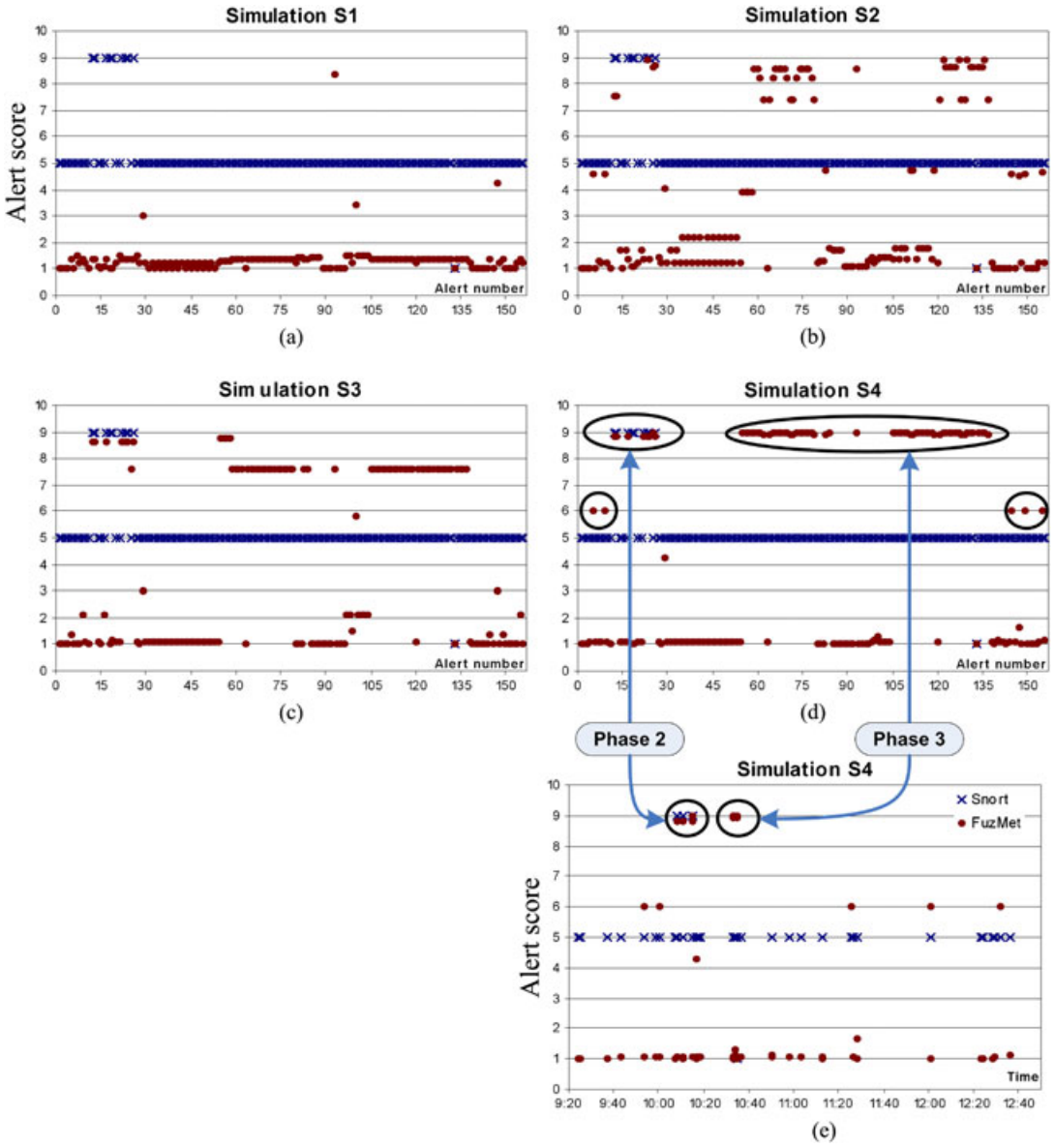

Figure 7. FuzMet configuration results

Of the four identifiable phases of the DDoS attack, only S_4 manages to detect all of them with a strong precision for phases 2 and 3 and a high-to-medium precision for phases 1 and 5 (phase 4 is only detectable by a host IDS and as such isexcluded from the evaluation). In contrast, SNORT manages to identify only phase 2. Figure 7(e) plots the results of S_4 based on time rather than on the number of alerts and shows the concentration of high-score alerts for phases 2 and 3, which last 09:03 and 01:51 minutes respectively. S_4 records 58 high-score alerts for phase 3, within an interval of 01:49 minutes. This interval is almost equal to the exact duration of phase 3 of the attack, which corresponds to the exploitation ofthe Sadmind vulnerability. It is also to be noted that while S_1 manages to generate only one high-score alert for phase 3, SNORT misses that phase completely and only detects phase 2 out of the four detectable phases. Overall, and based on the discovered results, S_4 represents the best configuration set.

However, it is difficult judge the optimal configuration set owing to the number of parameters, which may vary from one environment to another. Therefore we can use the discovered parameter

DOI: 10.1002/nem

set which led to the best results as a default configuration set.Because of the high number of configuration parameters required by the different metrics and fuzzy logic engine, this paper particularly emphasizes the problem of best configuration set for FuzMet with respect to the chosen dataset. In fact, the divergent behavior of FuzMet depending on how it is configured helped in identifying the configuration which led to best performance wherein all attack phases were identified.

10. IMPLICIT ATTACK GRAPH GENERATION

The attack graph represents the scenario that the attacker has taken from the first step until achieving the final goal. There have been some efforts in building the attack graph through correlating the alerts generated by the IDSs [5,10,27,28]. The attack graph provides the security analyst with a high-level representation of the attacker strategies. This is useful for the security analyst in order to figure out the steps of an attack from its first step to its final goal. Another advantage is related to the automatic exclusion of false positive alerts during attack graph construction since they are rarely correlated with the real attacks.

In this section, we show that our technique in computing the relationship metric is equivalent to an implicit construction of an attack graph. The relationship between alerts is used to generate the attack graph of Figure 8 for the studied dataset. The nodes represent the alert name as used by SNORT and the arrows represent the relationship between the alerts. Figure 8(a) shows the attack graph result in setting the time window parameter (RW) to 10 min. As we discussed earlier, this time interval parameter has a direct implication on the number of alerts investigated per each alert by the relationship metric. In Figure 8(a), the attack relationship threshold Th_r is set to 0.75, which means that a link exists from attack a_i to a_j only if $R(a_i, a_j) \geq 0.75$. As can be seen, the first phase of the attack (scanning) has been isolated from the rest ofthe steps. The reason behind this is that the period between the end of the first phase and the beginning of the second phase exceeds the time limit interval of 10 min. The relationship between the second and third phases is, however, successfully detected.

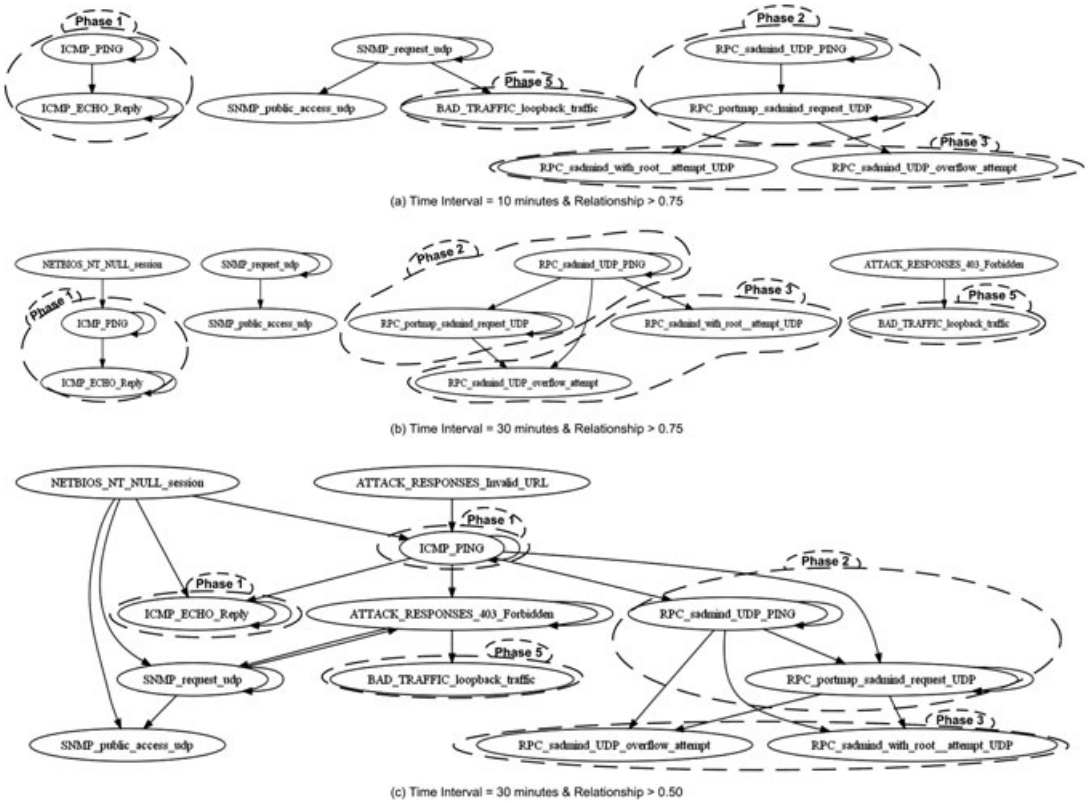

Figure 8. Implicit Attack Graph Generation

DOI: 10.1002/nem

Figure 8(b) shows the generated attack graph for an RW = 30 min with $Th_r = 0.75$. The first phase is still isolated from the second one due to the low follow-up probability (probe followed by a different probe). However, the second and third phases now have a stronger link.

Due to the high threshold value of the relationship score ($Th_r = 0.75$), most of the false positive alerts did not appear in the first and second attack graphs. In addition, those which appeared are isolated from the rest of the graph such as 'SNMP_request_udp' and 'SNMP_public_access_udp'.

In Figure 8(c), Th_r is lowered to 0.5 and RW is kept at 30 min. The generated graph includes a larger number of false alerts and links. However, the relationship between the first and the second phases is now captured, which can be valuable to the security analyst.

In summary, larger RW values help in better detecting relationships between alerts at the expense of longer computation times, while lower values of Th_r help in better detecting the actual relationships between all attack phases at the expense of a higher noise of false relationships. The security analyst can tune these parameters depending on the attack circumstances as no single values can fit all possible attack graph scenarios. However, since the attacks in an attack graph are definitely related, there are always ranges of values for RW and Th_r which will capture the actual attack graph, although with some expected noise of false positives.

11. CONCLUSIONS AND FUTURE WORK

This paper presented the specification and configuration issues of FuzMet, a system that uses fuzzy logic inference to evaluate and prioritize IDS alerts based on several metrics. We defined six metrics related to the applicability of the alert, importance of the target, sensor status, alert severity score, service vulnerability, and alerts static relationships. Because of the high number of configuration parameters required by the different metrics and fuzzy logic engine, the problem ofgood configuration for FuzMet was emphasized. The simulations were specifically conducted in order to determine such a configuration.

In the experimentation, the FuzMet approach has been applied to the alerts generated by SNORT with its maximum detection capability and using the DARPA 2000 LLDOS 1.0 dataset. The dataset features a DDoS attack on a remote site. Two hundred simulations were conducted and selected results were presented. While SNORT detected only one phase of the four detectable phases of the attack, the best found configuration of FuzMet managed to detect all of those phases with a good precision for two of the phasesand a very good precision for the others. Unexpectedly, even the worst case configuration of FuzMet also did well, in the sense that it managed to raise one single high score alert for a phase which SNORT did not detect at all.

The conducted simulations showed the viability of the FuzMet approach, at least for the selected intrusion scenario. In addition, the divergent behavior of FuzMet depending on how it is configured helped in identifying the configuration which leads tobest performance wherein all attack phases are identified. Overall, FuzMet proved to bring a concrete enhancement to the intrusion detection accuracy of SNORT, which is further enhanced when rescoring is applied.

However, even with the obtained results, we did not completely solve the configuration problem of FuzMet. In fact, many of the metrics were not configurable owing to the absence of their related data from the chosen scenario dataset. In addition, it is not possible to prove that the identified best simulation is the actual optimal configuration; or even whether it is the best just for that particular dataset. The search space involves a considerable number of parameters and further analytical analysis is required. As a future work, we plan to consider the use of FuzMet for real attack scenarios, investigate its usefulness for anomaly-based IDS alerts, and provide more analytical study of the configuration problem of the identified metrics as well as that of the fuzzy logic rules and inference engine.

ACKNOWLEDGEMENTS

This research was partially supported by the Natural Science and Engineering Council of Canada (NSERC) under its discovery program and partially by WCU (World Class University) program through the Korea Science and Engineering Foundation funded by the Ministry of Education, Science and Technology (Project No. R31-2008-000-10100-0).

DOI: 10.1002/nem

REFERENCES

1. Alsubhi K, Al-Shaer E, Boutaba R. Alert prioritization in intrusion detection systems. In IEEE Network Operations and Management Symposium, 2008; 33–40.
2. Snort. Available: http://www.snort.org [27 August 2011].
3. Li W, Zhi-tang L, Jie L, Yao L. A novel algorithm SF for mining attack scenarios model. In Proceedings of the IEEE International Conference on e-Business Engineering, 2006; 55–61.
4. Porras PA, Fong MW, Valdes A. A mission-impact-based approach to infosec alarm correlation. In RAID 2002: Recent Advances in Intrusion Detection: 5th International Symposium, Zurich, Switzerland, 2002.
5. Qin X, Lee W. *Statistical Causality Analysis of INFOSEC Alert Data*. Springer: Berlin, 2003.
6. Yu J, Reddy YVR, Selliah S, Kankanahalli S, Reddy S, Bharadwaj V. *TRINETR: An Intrusion Detection Alert Management System*. IEEE Computer Society: Washington, DC, 2004; 235–240.
7. Ranum MJ. *False Positives: A Users Guide to Making Sense of IDS Alarms*. ICSA Labs: Mechanicsburg, PA, 2003.
8. Debar H, Wespi A. *R&D, Aggregation and Correlation of Intrusion-Detection Alerts*. Springer: Berlin, 2001.
9. Julisch K. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security* 2003; **6**: 443–471.
10. Ning P, Cui Y, Reeves DS. Constructing attack scenarios through correlation of intrusion alerts. In Proceedings of the 9th ACM Conference on Computer and Communications Security, 2002; 245–254.
11. Valdes A, Skinner K. Probabilistic alert correlation. In RAID 2001: Recent Advances in Intrusion Detection: 4th International Symposium, Davis, CA, 2001.
12. Valeur F, Vigna G, Kruegel C, Kemmerer R. Comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing* 2004; **1**(3): 146–169.
13. Li Z, Chen Y, Beach A. *Towards Scalable and Robust Distributed Intrusion Alert Fusion with Good Load Balancing*. ACM Press: New York, 2006; 115–122.
14. Cuppens F, Ortalo R. LAMBDA: a language to model a database for detection of attacks. In *Recent Advances in Intrusion Detection: Third International Workshop*. Springer-Verlag London, UK, 2000.
15. Curry D, Debar H. Intrusion detection message exchange format (IDMEF). RFC 4765, 2007.
16. Staniford-Chen S, Tung B, Schnackenberg D. The common intrusion detection framework (CIDF). Information Survivability Workshop, Orlando, FL, 1998.
17. Feinstein B, Matthews G. The intrusion detection exchange protocol (IDXP). RFC 4767, 2007.
18. *National Vulnerability Database*. National Institute of Standards and Technology: Gaithersburg, MD.
19. Bugtraq. Available: http://www.securityfocus.com/archive/1 [27 August 2011].
20. Axelsson S. The base-Rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security* 2000; **3**(3): 186–205.
21. Abedin M, Nessa S, Al-Shaer E, Khan L. Vulnerability analysis for evaluating quality of protection of security policies. In QoP '06: Proceedings of the 2nd ACM workshop on Quality of protection, 2006.
22. Nguyen HT, Walker E. *A First Course in Fuzzy Logic*. CRC Press: Boca Raton, FL, 2006.
23. Yager RR, Zadeh LA. *An Introduction to Fuzzy Logic Applications in Intelligent Systems*. Kluwer: Norwell, MA, 1992.
24. Wang LX, Mendel JM. Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. *IEEE Transactions on Neural Networks* 1992; **3**(5): 807–814.
25. 2000 darpa intrusion detection scenario specific datasets. Lincoln Laboratory, MIT, Cambridge, MA, 2000.
26. Fuzzy Logic Toolbox. Available: http://www.mathworks.com/products/fuzzylogic [27 August 2011].
27. Ning P, Cui Y, Reeves DS, Xu D. Techniques and tools for analyzing intrusion alerts. *ACM Transactions on Information and System Security* 2004; **7**(2): 274–318.
28. Noel S, Robertson E, Jajodia S. Correlating intrusion events and building attack scenarios through attack graph distances. In 20th Annual Computer Security Applications Conference, 2004; 350–359.

AUTHORS' BIOGRAPHIES

Khalid Alsubhi received the bachelor degree with a first honor degree from the faculty of Computing and Information Technology at King Abdulaziz University in 2003. He received the Masters degree (MMath) in Computer Science from the University of Waterloo in 2008. He is currently a PhD student in David R. Cheriton School of Computer Science at the University of Waterloo.

Issam Aib received the MSc. and PhD. degrees in Computer Science from the University of Pierre & Marie Curie, Paris, France, in 2002 and 2007 respectively. He is currently a Postdoctoral fellow at the school of computer science of the University of Waterloo (Canada) where he is conducting research on policy-based and businessdriven management of networks and distributed systems since 2005. He is the recipient of the best student-paper award of the tenth IFIP/IEEE International Symposium on Integrated Network Management (IM 2007) for his work on the optimization of policy-based management.

DOI: 10.1002/nem

Raouf Boutaba received the MSc and PhD degrees in computer science from the University Pierre & Marie Curie, Paris, in 1990 and 1994, respectively. He is currently a professor of computer science at the University of Waterloo. His research interests include network, resource and service management in wired, and wireless networks. He served as the founding editor in chief of the IEEE Transactions on Network and Service Management (2007–2010) and on the editorial boards of several other journals. He has received several best paper awards and other recognitions such as the Premiers Research Excellence Award, the IEEE Hal Sobol Award in 2007, the Fred W. Ellersick Prize in 2008, the Joe LociCero award and the Dan Stokesbury in 2009.

DOI: 10.1002/nem