INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt (2014)
Published online in Wiley Online Library (wileyonlinelibrary.com) DOI: 10.1002/nem.1857

An efficient approach to reduce alerts generated by multiple IDS products

Tu Hoang Nguyen,[1,2] Jiawei Luo[1,*†] and Humphrey Waita Njogu[3]

[1]*College of Information Science and Engineering, Hunan University, Changsha, China*
[2]*Centre for Informatics and Foreign Language, Hanoi University of Industry, Hanoi, Vietnam*
[3]*Kenya Institute for Public Policy Research and Analysis (KIPPRA), Nairobi, Kenya*

SUMMARY

Intrusion detection systems (IDSs) often trigger a huge number of unnecessary alerts. Managing the overwhelming number of alerts, especially from multiple IDS products, is a concern to every security analyst. Analyzing and evaluating these alerts is a difficult task that frustrates the effort of analysts. In fact, true alerts are usually buried under heaps of false alerts. We have identified several research gaps in the existing alert management approaches that need to be addressed, especially when handling alerts from different IDS products. In this work, we present an efficient alert management approach that reduces the unnecessary alerts produced by different IDS products using two main modules: an enhanced alert verification module that validates alerts with vulnerability assessment data; and an enhanced alert aggregator module that reduces redundant alerts and presents them in the form of meta alerts. Finally, we have carried out experiments in our test bed and recorded impressive results in terms of high accuracy and low false positive rate for multiple IDS products. Copyright © 2014 John Wiley & Sons, Ltd.

Received 19 November 2012; Revised 4 December 2013; Accepted 10 February 2014

1. INTRODUCTION

Increasing cybercrime has seen a great demand for security devices for computer networks. Unfortunately, there is no single device with the ability to tackle all the network security concerns. As a result, many organizations are increasingly looking for additional security technologies to counter risk and vulnerability that other security tools fail to address. Intrusion detection systems (IDSs) are commonly used to complement other security tools in detecting network intrusions [1,2]. Recently, IDSs have gained wide acceptance as a valuable investment in organizations because they offer a layer of defense to computer networks.

Generally, IDSs gather and analyze information in a network in order to identify possible security breaches and generate an alert or alarm if an intrusion is detected. There are two classes of IDS [3]:

- Signature-based IDS: this recognizes patterns of attack and works in a similar way to antivirus software. The IDS essentially contains attack descriptions or signatures and matches them against the audit data stream, looking for evidence of known attacks. It employs signature databases of well-known attacks, and a successful match with current input raises an alert. Signatures generally target widely used applications or systems for which security vulnerabilities are widely advertised.
- Anomaly-based IDS: this looks for deviations from normal usage behavior in order to identify abnormal behavior. An anomaly detection technique relies on models of the normal behavior of a network. The IDS may focus on the users, the applications or the network. Behavior profiles

*Correspondence to: Luo Jiawei, College of Information Science and Engineering, Hunan University, Hunan, China
†E-mail: luojiawei@hnu.edu.cn

are built by performing statistical analysis on historical data, or by using rule-based approaches to specify behavior patterns.

IDSs are designed to generate alerts of high quality to the analysts; unfortunately, traditional IDSs have not lived up to this objective because they trigger an overwhelming number of unnecessary alerts [4]. In fact, most of these alerts are primarily irrelevant resulting from non-existing intrusions [3–5]. The art of detecting intrusions is still far from perfect [4,5]. Actually, the current IDSs appear ineffective due to large volumes of alerts. It is interesting that false alerts form the largest proportion of the total number of alerts produced by IDSs. The interesting alerts are often missed in the analysis because they are buried under heaps of large numbers of false alerts [6,7]. In busy and large networks, the analysis of alerts becomes an unmanageable task and hence it is difficult to understand the intrusions behind the alerts. It is important to note that reduction of alerts and provision of quality alerts are very critical aspects of alert management.

Our work focuses on reduction of the unnecessary alerts that are often generated by multiple signature-based IDSs. Alert reduction is a well-explored topic in alert management. More particularly, alert verification has been cited as a key component in alert management and looks to be very promising technique to deliver alerts of high quality. Generally, the alert verification process uses vulnerability assessment data to establish whether a given alert represents a real threat to the network or not. This technique filters out any alert that does not have a corresponding vulnerability in a given network. Although several efforts have been made in the literature regarding use of the alert verification technique to improve the quality of alerts, there are several research gaps that need to be addressed in order to reduce unnecessary alerts significantly, especially in a network with multiple IDS products from different vendors. As seen in Section 2, it is evident that knowledge-based approaches suffer from the issue of having incomplete details concerning vulnerabilities such as reference i.d.s that uniquely identifiers different vulnerabilities. This problem worsens when dealing with alerts generated by different IDS products because each product uses its own set of reference i.d.s, which may be different with other IDS products. In fact, the alert verification-based approaches do not have a comprehensive and effective collaborative architecture to improve the quality of alerts. We also noted that the act of alert verification may not guarantee final alerts of high quality. As noted in Section 2, little attention is given to alerts after the alert verification process. The validated alerts may contain a massive number of redundant and isolated alerts that need to be reduced. The trend of the multi-step intrusions is on the rise, leading to unmanageable redundant alerts. For example, a single intrusion can generate several alerts with common features. The analysis of single redundant alerts provides partial information on the attack that does uncover the real patterns of the attacks. This calls for a better post-alert verification mechanism that aggregates the individual redundant and isolated alerts representing every step of attack to have the ‘big picture’ of attacks for different IDS products.

This paper proposes an effective alert management approach that handles alerts generated by signature-based IDSs from different vendors. The approach has two main modules: the alert verification module, which validates alerts with vulnerability data; and the alert aggregation module, which reduces the volumes of redundant and isolated alerts belonging to the same attack activity within a particular time window for the different IDS products and present alerts in the form of meta alerts. With the experiments, we are able to demonstrate the effectiveness of the proposed approach. We are able to successfully reduce the unnecessary alerts from different IDS products. The contributions of this work are summarized as follows:

- Development of an effective alert verification-based collaborative architecture to improve the quality of alerts raised by multiple IDS products. This includes the development of an enhanced alert verification module that is able to successfully validate the alerts produced by multiple IDS products, and development of an enhanced alert aggregation module that aggregates alerts of multiple IDS products in order to reduce the high number of redundant and isolated alerts after the alert verification process. The aggregation module is able to further reduce similar alerts generated by different IDS products.
- Construction of comprehensive vulnerability assessment data in order to improve the accuracy and quality of alerts. The vulnerability assessment data is drawn from four sources: scan reports produced by different vulnerability scanners; popular known vulnerability databases such as

CVE and OSVDB; reference details from different IDS products; and the details of network resources of a given network.

The rest of the paper is organized as follows. Section 2 discusses related work. In Section 3 we discuss our proposed approach. Section 4 describes the experimental set-up and analysis of the results. Section 5 concludes the paper.

2. RELATED WORK

IDSs often do not generate alerts of high quality, for several reasons [4]. For example, IDSs use their default set of signatures and hence are prone to trigger alerts for most intrusions regardless of success or failure to exploit vulnerabilities in a given network. There are several existing approaches designed to improve the quality of alerts and they have their merits and demerits. For example, tuning and reconfiguring of the signature database may eliminate some of the false alerts but may not work well in a large network because it is difficult to have all sensors tuned to an acceptable false positive level. In addition, the process of tuning and reconfiguring databases requires extensive knowledge and experience of IDS signatures [3,7,8]. Moreover, the improper manipulation of signatures could lower detection rates, hence exposing critical network resources to risk, especially if the critical intrusions are not detected. Correcting all the known vulnerabilities before damaging intrusions take place in order to avoid alerts may not be a possible solution because some vulnerabilities are protocol based and thus an immediate patch may not be available [9].

Kruegel *et al.* [7] observed that most IDSs either do not either consider network context or do not fully understand what is being protected, and hence trigger volumes of meaningless alerts. In the field of alert management, the alert verification technique has been cited as a critical component in improving the quality of alerts. The underlying principle of alert verification is to filter out alerts that do have corresponding vulnerabilities in a given network, thereby improving the accuracy of alerts. It is argued that many of the false alerts can be reduced by focusing on the vulnerabilities of the protected network [7,8,10]. Valeur [1] notes that failure to exclude the alerts that refer to failed attacks may lead to false positive alerts being misinterpreted or given undue attention. The authors note that when any scheme receives false positives as input the quality of the results can be degraded significantly. Al-Mamory and Zhang [11] note that most of the general alert aggregation approaches (e.g. [12–17]) do not make full use of the information that is available on a given network. This means that the general alert correlation approaches usually rely on the basic information presented on alerts which may be inadequate and unreliable and may lead to poor correlation results. In fact, correlating alerts that refer to failed attacks can easily result in the detection of whole attack scenarios that are non-existent. It is crucial to integrate the network context information in alert correlation in order to identify the exact level of threat that the protected systems are facing. In this section, we describe several prominent works aimed at reducing unnecessary alerts. First we focus on the general alert verification-based approaches and later on the alert verification-based approaches that use collaborative architecture to improve the quality of alerts. In this section, we have compared our approach to other similar approaches while pointing out the key differences.

An interesting approach to verify alerts is proposed by Kruegel *et al.* [7] in order to improve the false positive rate of IDSs. The approach has architecture for alert analysis and generation of a prioritized report for security analysts. Basically, the work introduces a plug-in for Snort to verify alerts. That is, the plug-in integrates the Nessus vulnerability scanner into the Snort. When an alert is raised, it is not immediately forwarded to the analyst but is passed to the verification engine. The underlying assumption of this approach is that every Snort signature comes with a unique identifier to check the presence of a corresponding vulnerability. The approach looks very promising but lacks useful additional information on host architecture, software and hardware to support better alert verification. Again, the approach does not offer a collaborative architecture to handle alerts from multiple IDS products. In addition, the approach does not address the issue of reducing the redundant and isolated alerts after alert verification.

Some notable efforts have been done on aggregating alerts in order to have meta alerts; such works include Julisch [12], Valdes and Skinner [13], Debar and Wespi [14], Al-Mamory and Zhang [15], Sourour *et al.* [16] and Jan *et al.* [17]. These works have reported considerable progress in reducing unnecessary alerts. However, our approach employs an alert validation technique before the aggregation of alerts is done. It is important to note that most of the approaches that aggregate alerts usually deal with non-validated alerts (unverified alerts).

Ning *et al.* [18] propose an alert correlator on the basis of the observation that most attacks consist of several related stages, with the early stages preparing for the later ones. Hyper-alert correlation graphs are used to represent correlated alerts in an intuitive way. The approach is useful, but it is ineffective when the attackers use either a different or spoofed IP source address at each attack step.

A possible correlation approach based on integration of Snort, Nessus and Bugtraq databases is proposed by Massicotte *et al.* [19]. It is based on the reference numbers (identifiers) found on Snort alerts in order to link signature vulnerabilities contained in common vulnerability exposure (CVE), Bugtraq and Nessus scripts. The approach appears promising in trying to show a possible correlation based on reference numbers, but it is not effective because not all alerts have reference numbers and there is no guarantee that the lists provided by CVE and Bugtraq contain a complete listing of all vulnerabilities. In addition, unlike our approach, the approach does not incorporate a collaborative architecture to handle alerts from multiple IDS products.

Porras *et al.* [20] propose an alert filtering-based approach known as M-Correlator. It is based on knowledge of the network architecture and vulnerability requirements of different incident types in order to tag alerts with a relevance metric and then prioritize them accordingly. In this approach, any alerts representing attacks against non-existent vulnerabilities are discarded. The approach correlates security alerts produced by spatially distributed heterogeneous information security (INFOSEC) devices. The basic principle of this approach is that it takes into account the topology and operational objectives of the protected network when alerts are being correlated and finally evaluates the impact of alerts on the overall mission that a network infrastructure supports. There are two main processes in M-Correlator: correlation and aggregation. That work processes alerts from different sources such as IDS, firewalls and other devices, while our work deals with IDS alerts only. Unlike in our work, the authors do not propose a comprehensive vulnerability data model to fully integrate the vulnerability data into the network context.

A formal treatment of the integration of context information with alert messages known as M2D2 is proposed by Morin *et al.* [10]. The underlying principle of the approach is to use reference numbers to locate which vulnerability it is and then verify its genuineness using other context information. The approach performs alert correlation using different types of information. Based on these formally defined concepts, the approach examines the relationship between them and performs alert correlation. M2D2 mainly focuses on alert aggregation while concepts such as building the attack scenarios are still to be explored. Like the aforementioned approaches, this approach suffers because not all alerts have reference numbers and there is no guarantee that the list provided by either CVE or Bugtraq contains a complete listing of all vulnerabilities. In a more recent work, Morin *et al.* [21] present a data model known as M4D4, which formalizes the concepts and relationships of different network elements (such as network topology, network resources, vulnerabilities and attacks) in order to reason about alerts. The authors observe that many alerts (especially false positives) involve actors that are inside the monitored information system and whose properties are consequently also observable. The two aforementioned approaches do not address the issue of redundant alerts after the alert verification process.

Hubballi *et al.* [3] propose a false positive alert filter to reduce false alerts. The underlying principle of this approach is the construction of a threat profile of the network which is used for alert correlation. This approach compares the IDS alerts with network-specific threats and finally filters out the unnecessary alerts. Basically, the approach has two major steps: (i) alerts are compared with the threat profile in order to generate correlation binary vector generation; and (ii) classification of correlation binary vectors using neural networks. We wish to note that the idea of comparing the alerts with the threat profile (vulnerabilities) appears promising in improving the accuracy of the final alerts. Nevertheless, the approach does not incorporate a collaborative architecture to handle alerts from multiple IDS products. In addition, the approach does not address the issue of reducing the number of similar and repetitive alerts after verification.

Numerous works have proposed to evaluate alerts using some metrics that are computed based on vulnerability data. For instance, Bakar and Belaton [22] propose an intrusion alert quality framework (IAQF) to improve the quality of final alerts. The underlying principle of this framework is the use of vulnerability information that forms the basis to compute alert metrics (such as accuracy, reliability, correctness and sensitivity). Such a metric prepares alerts for higher-level reasoning and thus better decision making. The proposed framework has the following modules: alert collection, host/network information gathering, quality criteria scores measurement and normalization. There are also storage databases that store different types of data, i.e. raw alerts, host/network data, quality criteria rules, and enhanced and enriched alert data. We note that the framework improves the alert quality before alert verification but does not take into consideration the issue of reducing the redundant and isolated alerts after alert verification. In addition, the approach mentions the concept of alert correlation but has not implemented it. Similarly, Njogu and Jiawei [23] propose an approach to reduce unnecessary alerts that uses vulnerability data to compute alert metrics. This approach computes the similarity of alerts based on the alert metrics. All the alerts that show close similarity are grouped together into one cluster. An extension of this work is proposed by the same authors [24] in order to reduce unnecessary alerts based on enhanced vulnerability assessment data. The authors use the vulnerability assessment data to filter out any alert with no corresponding alerts, thus improving the accuracy and quality of alerts. The aforementioned approaches have reported tremendous progress in reducing unnecessary alerts; however, the approaches do not incorporate an effective collaborative architecture to handle alerts from multiple IDS products. Our new work seeks to strengthen our previous work in four ways: the first is to improve the accuracy of alerts by incorporating reference details of multiple IDS products in the vulnerability data; second is the use of dedicated alert verifiers for every IDS product; third is the use of dedicated alert aggregators for every IDS product; and finally we propose an approach that can aggregate similar alerts generated by multiple IDS products.

We noted that there are several approaches that use a collaborative technique in order to reduce unnecessary alerts. An interesting collaborative framework architecture known as TRINETR is proposed by Yu *et al.* [25]. The proposed scheme is used for multiple intrusion detection systems in order to work together to detect real-time network intrusions. The proposed architecture is composed of three core parts: (i) collaborative alert aggregation; (ii) knowledge-based alert evaluation; and (iii) alert correlation to cluster and merge alerts from multiple IDS products to achieve an indirect collaboration among them. We note that the approach has the ability to reduce false positives by integrating network and host system information into the evaluation process, but the approach has a major drawback. This approach has not implemented the alert correlation part, which is very important when generating condensed alert views known as meta alerts.

A collaborative and systematic framework to correlate alerts from multiple IDSs by integrating vulnerability information is proposed by Liu *et al.* [26]. The basic principle of the approach is to apply contextual information to distinguish between successful and failed intrusion attempts. The approach assigns confidence values to the alerts immediately after alert verification. The confidence values are 0 (for false alert) and 1 (for true alert). The corresponding actions are triggered based on the confidence values. We note that the approach has some merits but does not provide details of the procedure used to validate the alerts and does not include details on how the alerts are transformed into meta alerts. The proposed scheme is not able to differentiate levels of alert relevance. In addition, the approach only processes alerts with reference numbers (alerts without reference number are not considered), thus lowering detection rates. Similarly, an approach to filter innocuous attacks that takes advantage of the correlation between IDS alerts and the threat profile is proposed by Michele *et al.* [9]. Among the core units of the proposed approach are filtering and ranking units. A similar but extended work is presented by the same authors. They propose a distributed architecture [27] to provide security analysts with selective and early warnings. According to this approach, alerts are ranked based on match or mismatch of alerts between the alert and the vulnerability assessment data. We regard this type of ranking to be restrictive because: first, it relies on software match or mismatch to determine whether an alert is important or unimportant; Secondly, it does not indicate the degree of relevance of alerts and hence does not offer much help to the analyst. Moreover, the two aforementioned approaches do not consider the issue of reducing redundant and isolated alerts after alert verification. In our work, we prioritize alerts into different levels according to their degree of interestingness.

A general correlation framework that includes a set of components such as alert fusion, multi-step correlation and alert prioritization is proposed by Valeur *et al.* [1]. The raw alerts are processed in these components. In this approach, the alerts are fused before they are verified. Later, the alerts are forwarded to the alert correlation and prioritization components. The authors observe that reduction of alerts is an important task in alert management and note that when any scheme receives false positives as input the quality of the results can be degraded significantly. The framework proposed in our work is inspired by logical framework of Valeur *et al.* There are several key differences. First, the authors use scanning reports to verify alerts, whereas our work uses up-to-date and comprehensive vulnerability data to validate alerts. Secondly, we verify alerts prior to the aggregation phase process because efforts to improve the quality of alert should start in the early stages of alert management. Thirdly, the framework proposed by the authors has several components that may contribute to negative performance of the framework, whereas our approach has fewer but robust components. Lastly, our work focuses not only validating alerts reported by multiple IDSs but also aggregates alerts of multiple IDS products.

An architecture for automatic alert verification known as ATLANTIDES is proposed by Bolzoni *et al.* [28]. ATLANTIDES reduces false positives both in signature- and anomaly-based IDSs. It has an engine to correlate alerts with an output anomaly detector (OAD), which acts as an anomaly detector in the reverse channel. The underlying principle of the approach is that there should be an anomalous behavior seen in the reverse channel in the vicinity of the alert generation time. Therefore, if these two are compared a good guess about the attack corresponding to the alert can be determined. We note that the time window used to look for the alert correlation is very critical for correctness of the approach. Therefore, a very small time window may result in an increase of false positive alerts, hence affecting performance.

In summary, we observed that most of the alert verification-based approaches described in this section are able to validate alerts successfully. However, there are several issues that need to be addressed, as evidently seen in this section. This section has revealed that most of the vulnerability-based approaches are able to produce alerts that are useful in the context of the network. However, these approaches are still at the preliminary stage and there are some research gaps that need to be addressed in order to produce better results. The act of validating alerts may not guarantee alerts of high quality because the validated alerts may contain huge volumes of redundant alerts, as evidently seen in this section. Also as seen in this section, most of the existing approaches hardly process alerts after alert verification; hence the need to consider the issue of reducing the huge number of redundant and isolated alerts after verification. In addition, the vulnerability-based approaches focus on improving the quality of alerts that are generated by one particular IDS product and therefore do not offer a collaborative architecture to handle alerts from multiple IDS products. There is a need to focus on how to reduce the unnecessary alerts generated by multiple IDS products and therefore advance the notion of collaborative architecture for different IDS products. Lastly, we noted that several approaches rely on incomplete data such as incomplete reference i.d.s and outdated vulnerability assessment data to validate alerts and hence are likely to give inaccurate results. As noted earlier, our proposed approach has an architecture that employs the following modules: an alert verification module that uses vulnerability assessment data to validate alerts; and an aggregation module to merge the alerts of multiple IDS products in order to reduce the high number of redundant and isolated alerts after alert verification process.

3. THE PROPOSED APPROACH

In this section we describe the proposed architecture in order to address the challenges of alert verification-based approaches. Basically, the proposed solution has two major components: alert verification module and alert aggregation module. The proposed approach collects raw alerts produced by multiple IDS products using the alert pre-processor unit as illustrated in Figure 1. The function of the pre-processor is to collect raw alerts and pre-process the alerts.

Alert pre-processing involves converting the raw alerts into intrusion detection message exchange format (IDMEF) [29] and the extraction of important features of alerts such as IP and port numbers.

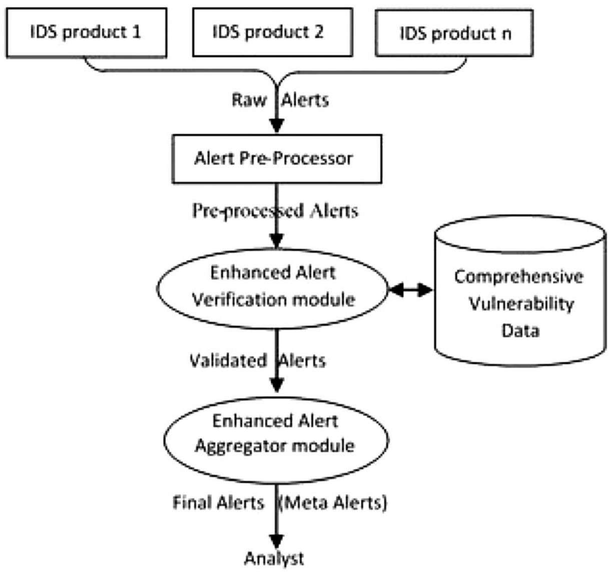
IDS product 1
IDS product 2
IDS product n
Raw Alerts
Alert Pre-Processor
Pre-processed Alerts
Enhanced Alert Verification module
Comprehensive Vulnerability Data
Validated Alerts
Enhanced Alert Aggregator module
Final Alerts (Meta Alerts)
Analyst

Figure 1. The general architecture of the proposed alert management approach.

The purpose of IDMEF is to standardize the formats of alerts generated by multiple IDS products, since different types of IDS products have different alert formats. The pre-processed alerts are forwarded to the alert verification module and then to the alert aggregation modules.

In summary, the main function of alert verification is to validate the alerts in order to filter out those with no corresponding vulnerability, thus improving the accuracy and quality of final alerts. The main function of the alert aggregation module is to reduce the redundant and isolated alerts from different IDS products. Eventually, the final alerts are presented to analysts in the form of meta alerts. Figure 1 illustrates the proposed strategy.

3.1. Enhanced alert verification module

As mentioned earlier, the signature-based IDSs are run with their default configurations (signature database). In fact, these IDSs are not fully integrated with network resources and therefore do not check the relevance of an intrusion (reported in the alert) to the local network context. As a result, the IDSs may generate huge volumes of raw alerts, the majority of which are irrelevant alerts and are not useful in the context of the network.

The alert verification module receives pre-processed alerts from the alert pre-processor. Unlike existing alert verification approaches [3,6,7,18,25] that depend on incomplete and incomprehensive vulnerability assessment data to process all the raw alerts, our work introduces comprehensive vulnerability assessment (CVA) data to improve the quality of alerts before they are forwarded to the security analysts. The enhanced alert verification module uses CVA data to validate the raw alerts produced by multiple IDS products. The verification module validates the raw alerts by measuring their similarity to corresponding vulnerabilities contained in CVA data, as illustrated in Figure 2. Both alert and vulnerabilities have comparable features and hence it is easy to measure the similarity. The

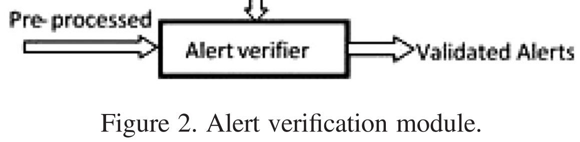

Figure 2. Alert verification module.

validation process helps to determine the seriousness of alerts with respect to the network under consideration. The uniqueness of our verification module is how we have constructed the CVA data, as described in Section 3.1.2. Unlike other approaches, our approach has implemented a simple but robust alert verification process.

From the literature it is noted that appropriate referencing of vulnerabilities plays a critical role in ensuring an accurate alert verification process. Different IDS products refer to the vulnerabilities differently, hence the need to standardize the referencing of vulnerabilities. In the next section, we explore possible solutions, such as the open source vulnerability database (OSVDB), to assist in referencing the vulnerabilities. The details of how a comprehensive list of vulnerabilities is derived from various sources of vulnerabilities are discussed in the next section.

In order to improve the performance of alert verification, as mentioned earlier, the alert verification module has dedicated alert sub verifiers for different classes of attack. The scope of the types of attack considered in this work is: DoS, Telnet, FTP, MySQL and SQL. The alert verifier has six sub verifiers, as follows (see Figure 3):

- DoS alert sub verifier: validates alerts reporting DoS attacks.
- Telnet alert sub verifier: validates alerts reporting Telnet attacks.
- FTP alert sub verifier: validates alerts reporting FTP attacks.
- SQL alert sub verifier: validates alerts reporting SQL attacks.
- MySQL alert sub verifier: validates alerts reporting MySQL attacks.
- Undefined alert sub verifier: validates alerts reporting attacks that have not been considered during the design of alert verification module (handles related alerts reporting new attacks and not included in the above five classes).

The proposed work introduces multiple alert sub verifiers to handle alerts regardless of the IDS product employed in the network. The main reason for using multiple alert verifiers in our approach is to improve the alert verification process, as shown in Figure 3. Another key benefit of using multiple sub verifiers is the ease of deployment in relatively large networks, especially where multiple products are deployed. As mentioned earlier, the main goal of the existing alert verification-based approaches as seen in Section 2 is to validate alerts from one particular IDS product. Unlike the existing approaches, the principal function of our alert verification module is to improve the accuracy of alerts (produced by multiple IDS products) by validating the alerts with vulnerability assessment data. The alert verification module validates alerts by measuring their similarity with respect to the network in order to determine the importance of alerts, as illustrated in Figure 4.

The alert verification process basically involves measuring the match or mismatch of features between alerts, as shown in Table 1, and vulnerability (vulnerability assessment data shown in Table 2)

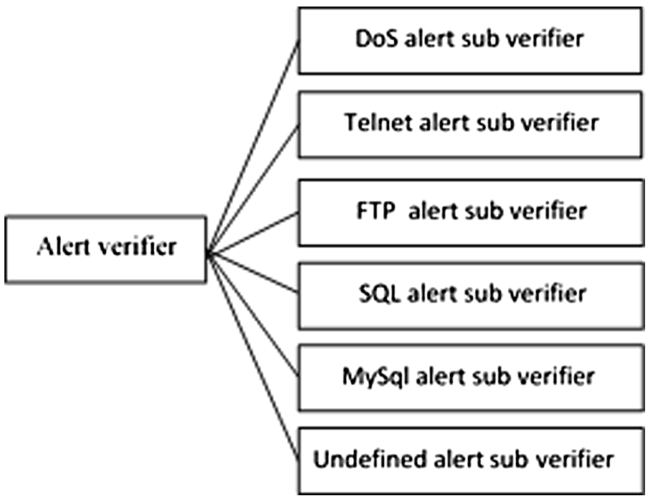

Figure 3. Sub verifiers of alert verification module.

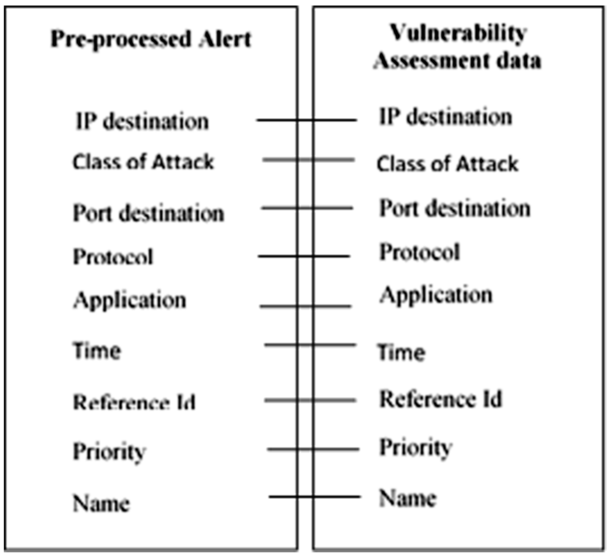

Figure 4. Comparing the features of alerts and vulnerability assessment data.

since both the alert and vulnerability assessment data have comparable features (refer to Figure 4). The vulnerability assessment data represent potential threats likely to be exploited by attackers and therefore the alert verification module improves the accuracy of alerts. The underlying assumption of the alert verification process is that, the higher the number of matches (alert relevance score), the higher is the likelihood of a successful intrusion. The alert relevance score used in our approach helps to separate false alerts from true alerts. It is worth noting that we performed a series of regrouping of alerts on the scale with the goal of searching for the best thresholds for the true and false alerts, as shown in Table 3. We categorize the alerts into three groups depending on the alert relevance score. As shown in the same table, the ideal relevant alert represents the most interesting alert, partial relevant alert represents the average interesting alert, while the non-relevant alert represents an alert that is not worth further investigation. We settled on the thresholds shown in the table but these values can be adjusted depending on the network environment.

3.1.1. Enhancing the semantics of alerts

Generally, the features of any alert provide low-level information; hence relying solely on this information may increase cases of important alerts being misinterpreted, ignored or delayed when they are being evaluated. In order to improve the semantics of alerts, the alert verification module places alerts into different categories based on the alert relevance scores as shown in Table 3. In a nutshell, the alert relevance is based on similarity of alerts and their corresponding vulnerabilities. Figure 4 shows how alerts are compared with vulnerability assessment data.

3.1.2. Construction of comprehensive vulnerability assessment data (CVA data)

Existing alert verification-based approaches [3,7,18,25] build vulnerability data that appear effective in validating the alerts from one IDS product. The existing approaches construct vulnerability assessment data from sources such as known vulnerability data and network resources and do not consider the aspect of including additional reference information from multiple IDS products. In our work, we have constructed CVA data in order to improve the quality of alerts that are generated by different IDS products. In brief, CVA data are comprehensive and effective data that represent the threat profile of the network. It lists all network-specific vulnerabilities by their reference i.d., name, priority, IP address, port, protocol, class, time and applications. We use the threat profile generator to construct CVA data in order to identify the relevant vulnerabilities representing actual current threats to the network. We design and implement a comprehensive vulnerability assessment data using the threat profile generator, as shown in Figure 5.

Table 1. Pre-processed alert snapshot

IDS Product	Reference i.d.*	Name*	IP_{dest}*	*$Port_{dest}$	IP_{src}	$Port_{src}$	Priority (severity)*	Protocol*	Class*	Time*	Application*
Snort	CVE-1999-0001	Teardrop	192.168.1.3	1238	192.168.1.1	1238	High	TCP	DoS	8:9:2012:14:16:02	Windows
Shoki	CVE-1999-0527	Telnet resolve host conf	192.168.1.2	1026	192.168.1.1	1026	Medium	TCP	U2R (Telnet)	8:9:2012:14:16:02	Telnet, Windows
Snort	CVE-2011-1965	Buffer overflow	192.168.1.4	80	192.168.1.1	80	Medium	TCP	DoS	8:9:2012:14:16:02	Windows

Key: *used in alert verification.

Table 2. CVA data snapshot

Reference i.d.	Name	IP address	Port	Priority(severity)	Protocol	Class	Time	Application
CVE-1999-0001	Teardrop	192.168.1.3	1238	High	TCP	DoS	8:9:2012:14:16:02	Windows
CVE-1999-0527	Telnet resolve host conf	192.168.1.2	1026	Medium	TCP	U2R (Telnet)	8:9:2012:14:16:02	Telnet, Windows
CVE - 2011-1965	Buffer overflow	192.168.1.4	80	Medium	TCP	DoS	8:9:2012:14:16:02	Windows

Table 3. Alert relevance score scale

Threshold	Alert type
7-9	Ideal relevant alerts
4-6	Partial relevant alerts
0-3	Non-relevant alerts

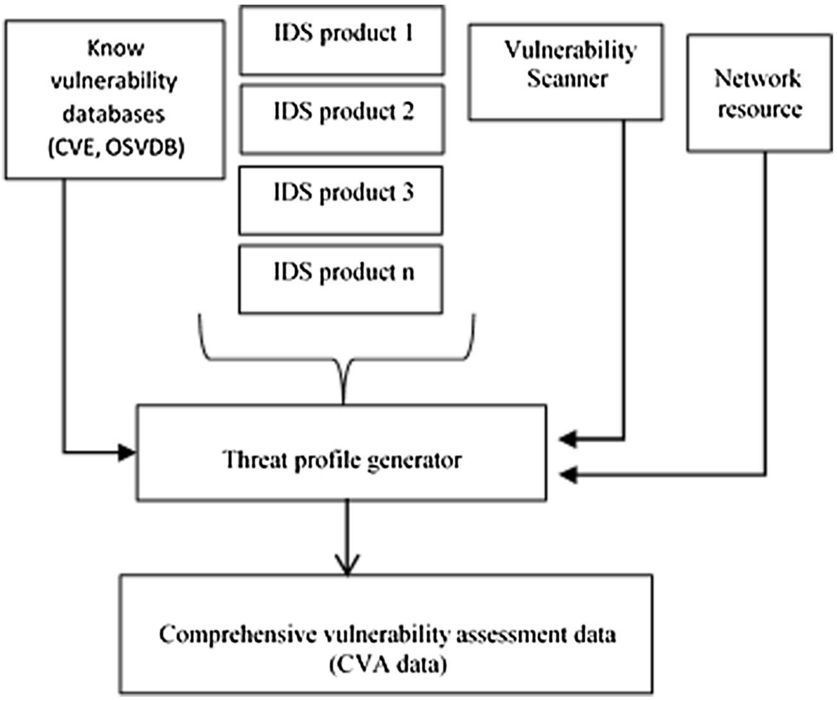

Figure 5. Construction of comprehensive vulnerability assessment data

Unlike other approaches described in Section 2, the generator of the proposed approach draws data from four sources: scan reports produced by different vulnerability scanners; popular known vulnerability databases such as CVE [30] and OSVDB [31]; reference details from different IDS products; and the details of network resources of a given network. The generator establishes appropriate relationships in the data drawn from the above sources.

As noted earlier, the appropriate referencing of vulnerabilities plays a critical role in ensuring accuracy in the alert verification process. In the literature it is reported that different IDS products refer to the vulnerabilities differently; hence the need to standardize the referencing of vulnerabilities. In order to address this issue, researchers have proposed possible solutions such as the OSVDB to assist in referencing the vulnerabilities. Such solutions may address this issue to some extent but may not contain a comprehensive list of reference details of all the vulnerabilities. This makes it difficult to have an independent alert verification based on one vulnerability database. Therefore, in our approach, we complement the OSVDB database with reference details from multiple IDS products; hence the need for a verifier for each product. We have automated the process of picking non-referenced vulnerabilities from different IDSs in order to build comprehensive CVA data. Additional information on building the CVA data is given in Section 4.

In the literature it is documented that dealing with attack identifiers may pose a challenge, especially when dealing with alerts containing different names from those known by the vulnerability databases. The IDS vendors may use different names for the same attack and so far they have not come to an agreement on standardizing the naming of alerts. In order to address this challenge, we include the aspect of how different IDS products reference the vulnerability (such as reference i.d.s) into the CVA data in order to have a more accurate and comprehensive list of vulnerabilities. In addition, the alert verifiers are trained adequately using CVA data on how to perform the alert verification with high accuracy. CVA data are so exhaustive and comprehensive that they permit the completeness of the attack identifiers for our test bed.

Steps of alert validation. One of the functions of the alert verification module is to assign each alert to the attack class that might have produced it. The alert verifier makes this decision based on their attack identifier field (class of attack) in the alert. To facilitate the verification process, there are six sub verifiers that are used to process alerts regardless of the IDS products. The procedure for validating alerts using the IDS Alert-CVA data verifier is illustrated below:

Input: pre-processed alerts, CVA data
Output: validated alerts (with alert relevance score)
Step 1: compare features of pre-processed alert with the corresponding vulnerabilities in CVA data in this order:

- IP
- ReferenceId
- Port
- Time
- Application
- Protocol
- Class
- Name
- Priority

Step 2: search for corresponding vulnerabilities from CVA data. Get Alert.IP_{dest} to extract the corresponding vulnerabilities in CVA data (CVA data.IP that match Alert.IP_{dest}).
Step 3: compute alert relevance score for every corresponding vulnerability and choose the one with the highest score.

- If Alert. IP_{dest} match CVA data.IP Then IP_Score=1 Else IP_Score=0
- If Alert.ReferenceId match CVA data. ReferenceId Then Reference_Score=1 Else Reference_Score=0
- If Alert.$Port_{dest}$ match CVA data.Port Then Port_Score=1 Else Port_Score=0
- If Alert.Time is greater or equal to CVA data.Time Then Time_Score=1 Else Time_Score=0
- If Alert.Application match CVA data.Application Then Application _Score=1 Else Application _Score=0
- If Alert.Protocol match CVA data.Protocol Then Protocol_Score=1 Else Protocol _Score=0
- If Alert.Class match CVA data.Class Then Class_Score=1 Else Class_Score=0
- If Alert.Name match CVA data.Name Then Name_Score=1 Else Name _Score=0
- If Alert.Priority match CVA data.Priority Then Priority _Score=1 Else Priority _Score=0
- Alert Relevance Score =IP_Score + Reference_Score + Port_Score + Time_Score + Application_Score + Protocol_Score + Class_Score + Name _Score+ Priority _Score

Step 4: categorize the alerts based on alert relevance score (see Table 3).
Step 5: alerts with scores of 4 and above (both ideal and partial relevant alerts) are forwarded to the alert aggregator.
Step 6: delete alerts with scores of 3 and below (alerts with little or no similarity with CVA data)

Note: For simplicity reasons, any match is awarded a value of 1, while a mismatch is awarded a value of 0.

To illustrate the concept of alert verification and how the alert metrics are computed, consider the following example. Table 1 shows a snapshot of pre-processed alerts. Table 2 shows a section of CVA data. In Table 1 the first alert reports a teardrop attack targeting a host with IP_{dest} 192.168.1.3 on $Port_{dest}$ 1238. Using the alert IP_{dest} 192.168.1.3, the alert verifier extracts all potential vulnerabilities in CVA data matching the IP address. In this case, CVA data have only one vulnerability matching the alert IP_{dest}. The verifier then matches other features (reference i.d., port, severity, protocol, class, time, name and application) of the alert against the said vulnerability. The alert score is computed as follows:

IP_Score = 1 because IP addresses are matching and so are the rest of the features; Reference_Score = 1; Port_Score = 1; Time_Score = 1; Application_Score = 1; Protocol_Score = 1; Class_Score = 1; Name_Score = 1; Priority_Score = 1. Therefore, the alert score (total number of matches) is 9.

The verifier enriches the alert by categorizing the alert as an ideal relevant alert because the alert has an alert relevance score = 9.

In summary, unlike other alert verification-based approaches [3,6,18,25] which focus on improving the quality of alerts generated by one IDS product, the proposed alert verification module is designed with the goal of improving the quality of alerts generated by multiple IDS products. The other key difference between our work and related studies is how the vulnerability data are constructed. We include the attack reference details of multiple IDS products in order to build comprehensive data.

3.2. Alert aggregation module

The literature has documented that the trend of the multi-step intrusions is on the rise and therefore contributing to the high number of unmanageable redundant alerts. Actually, a single intrusion can generate several alerts with common features. Generally, the analysis of single redundant alerts may provide partial information on the attack and hence is not valuable in uncovering the real patterns of the attacks. In order to address this issue, there is a need to aggregate the individual redundant and isolated alerts representing every step of the attack to have an overall view of attacks for the different IDS products. The trend of large modern networks is to implement multiple IDS products, especially where the network is viewed to be prone to attacks and has valuable information. The era of one IDS product monitoring a network segment is fading, especially where the information is viewed as important.

We designed the alert aggregation module with the goal of reducing the volumes of redundant alerts belonging to the same attack activity within a particular time window. The alert aggregation module receives validated alerts (unrelated, isolated and redundant alerts) from the alert verification module. As shown in Figure 6, the module has different sub aggregators for different classes of attacks. We used different dedicated sub aggregators in order to simplify the process of alert aggregation, representing different classes of attacks. These sub aggregators are adjusted dynamically and automatically according to each class of attack.

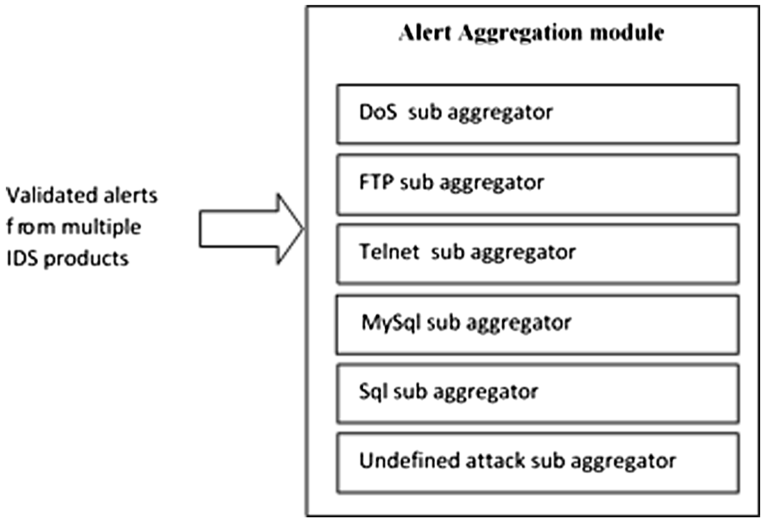

Figure 6. Alert aggregation module

The scope of the attacks considered in this work includes DoS, Telnet, FTP, MySQL and SQL. The aggregator has six sub aggregators, as shown in Figure 7:

- DoS alert sub aggregator: aggregates alerts reporting DoS attacks.
- Telnet alert sub aggregator: aggregates alerts reporting Telnet attacks.
- FTP alert sub aggregator: aggregates alerts reporting FTP attacks.
- SQL alert sub aggregator: aggregates alerts reporting SQL attacks.
- MySQL alert sub aggregator: aggregates alerts reporting MySQL attacks.
- Undefined alert sub aggregator: aggregates alerts reporting attacks that have not been considered during the design of the alert aggregation module (i.e. processes redundant alerts reporting new attacks and not covered in the other classes).

The underlying principle of the alert aggregation module is simple. Each alert sub verifier of a given class of attack forwards alerts to the corresponding sub aggregator of that specific class of attack. To further illustrate this, a DoS alert sub verifier forwards its alerts to DoS alert sub aggregator. We wish to note that the final alerts of the alert aggregation module are in the form of meta alerts (meta alerts contain summarized information of related alerts of a given intrusion).

Some definitions used for the variables of each meta alert M are as follows:

- Attack class of meta class (M.Attack_Class): denotes the class of attack of alerts.
- Relevance of meta alert (M.Rel): denotes the relevance of alerts.
- Create time of meta alert ($M.create_{time}$): denotes the create time of the meta alert and is derived from the stamp time of the first alert that created the meta alert.
- Total alerts ($M.Nbre_{alerts}$): denotes the number of alerts contained in the meta alert. This number is increased each time a new alert is fused to meta alert.
- Update time ($M.update_{time}$): denotes the time of the last update in the meta alert and represents the time of the most recent alert fused to the meta alert.
- Non-update time ($M.non\text{-}update_{time}$): denotes the time elapsed since the last update made on the meta alert. The time is reset to 0 each time a new alert is fused to the meta alert.
- Stop time ($M.stop_{time}$): denotes the time when the meta alert is assumed to have completely fused the necessary alerts.
- Time out ($M.Time_{out}$): denotes the time that a meta alert should continue waiting for additional alert (s); it varies with the class of attack.
- Exploit cycle time (EC_{time}): denotes the time that an exploit is believed to take (attack period); it varies with the class of attack.

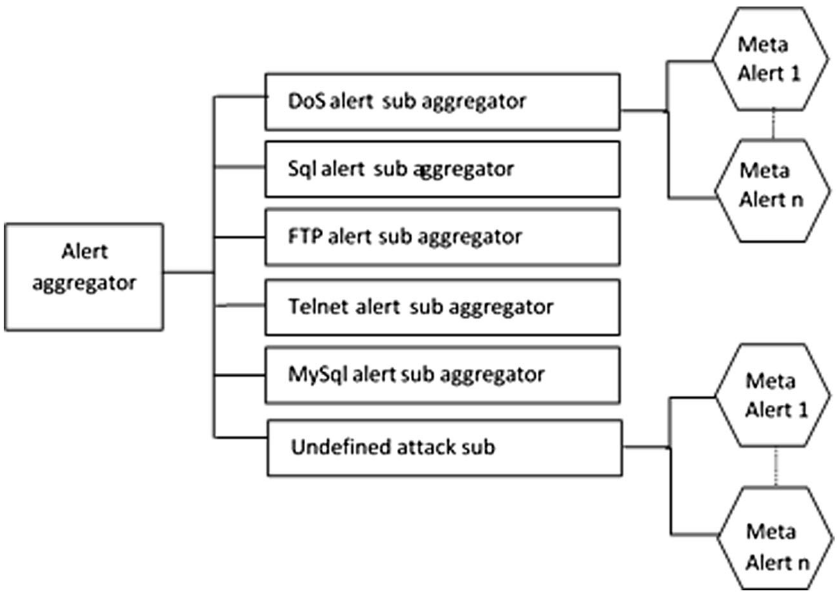

Figure 7. Sub aggregators of alert aggregator

- Additional information such as source address ($M.IP_{src}$ and $M.Port_{src}$) and destination address ($M.IP_{dest}$ and $M.Port_{dest}$).

The procedure of aggregating alerts is described as follows. The sub aggregator of a particular class of attack uses the IP address(es) of a given validated alert to identify the potential meta alerts. The sub aggregator measures the similarity of the validated alert and the existing meta alerts in order to identify the best meta alert representing the alert.

In this work, we only consider and finally fuse an alert that fully corresponds (all overlapping features are equal: IP, port, time) to an existing meta. This means that if an alert fully corresponds to a given meta alert then details of the meta alert are updated ($M.Nbre_{alerts}$ is incremented, $M.update_{time}$ is updated and $M.non\text{-}update_{time}$ is reset to 0). We wish to note that only one meta alert can be chosen from potential meta alerts and an alert can belong to only one meta alert. However, if the meta alert does not exist that fully corresponds to the alert being considered, then a new meta alert is created. The new meta alert draws basic details from that alert, i.e. $M.Nbre_{alerts}$ is set to 1, the timestamp of the alert becomes the $M.create_{time}$ and class of attack of alert becomes M.Attack_Class. It is very important to know how long an existing meta alert should remain active in order to deliver timely and accurate meta alerts to the analysts. To determine the activeness of meta alerts, we used the following policy:

- If $M.non\text{-}update_{time} < M.Time_{out}$, then the meta alert under consideration continues to wait for additional related alerts, and when the related alerts are received the meta alert details are updated.
- If $M.non\text{-}update_{time} \geq M.Time_{out}$, then M.stoptime is updated and the meta alert under consideration stops waiting for additional alerts.

In order to achieve the best results of alert aggregation, the values assigned to the $M.non\text{-}update_{time}$ and $M.Time_{out}$ differ according to the class of attack being considered. This helps to aggregate the necessary alerts appropriately. It is important to differentiate the functions of $M.Time_{out}$ and EC_{time}. The $M.Time_{out}$ ensures the necessary alerts are fused to the meta alert, while the EC_{time} ensures all the meta alerts reporting a particular attack are formed accordingly. The EC_{time} of a given class of attack helps to determine the extent of an attack using meta alerts generated regarding a particular attack within an exploit cycle time. We also note that each alert sub aggregator of a particular class of attack has its own $M.Time_{out}$ and EC_{time}. In this work, both $M.Time_{out}$ and EC_{time} of a given meta alert are not necessarily equal, for several reasons. The major reason is that $M.Time_{out}$ of a given meta alert could be lower than EC_{time} of the same meta alert. Generally, most of the events forming the different steps of intrusion are generated in the early stage of an exploit cycle [32] hence there is no need to wait for the complete exploit cycle timeframe of an attack. In addition, different types of network experience different types of attack. Some networks may be prone to both single-step and multi-step attacks. Therefore, it would be inappropriate for single-step attacks to have longer $M.Time_{out}$. This could lead to an unreasonable delay before analysts receive the meta alerts.

It is important to note that EC_{time} of a given attack may expire before all steps of the corresponding attack scenario are executed. This situation may appear like a major drawback but this is not so because the information collected on meta alert(s) at that time could show the analyst what the intruder has performed and obtained. Therefore the analyst is in a position to predict how the intruder will perform in future as well as think of how to stop future attacks. Our approach presents a big advantage because experienced analysts are able to adjust the values of $M.Time_{out}$ and EC_{time} of a given class of attack in order to optimize the detection of all attack steps. In addition, it is noted that a multi-step attack is likely to generate several meta alerts in a given exploit cycle time, whereas a single-step attack is likely to generate one meta alert within a given exploit cycle time.

The final step involves forwarding the meta alerts to the analysts. In this we assume that the alerts are forwarded immediately to the alert aggregation module by the alert verification module.

The proposed solution helps to identify the nature of the attack since it is implemented to monitor network traffic. Attack traffic could be directed to a number of hosts in a network; thus the solution helps to identify the nature and extent of the attacks.

3.2.1. Aggregation of meta alerts

It is important to note that aggregating alerts into meta alerts may not be a completely effective way to reduce unnecessary alerts. It is infeasible to manually process the meta alerts, particularly those that are sharing a number of features. In fact, when a given attack occurs in a network (with multiple IDS products), different IDS products may respond by generating alerts that may have some similar features such as source address, destination address and reference i.d.s. Such meta alerts need to be aggregated further based on their common features in order to reduce unnecessary alerts, as illustrated in Figure 8. For example, when a particular attack occurs in a given network, an IDS product may generate an alert that may look similar to an alert generated by another IDS product (if the two IDS products are network IDSs and have been installed in the same network).

The meta alert aggregator further processes the meta alerts and give output in form of aggregated meta alerts. Aggregating meta involves merging the meta alerts that share some features such as reference i.d.s, source and destination addresses within a class of attack. In a nutshell, aggregating meta alerts reduces overall computational costs and analysts are likely to take a relatively shorter time to review the final alerts.

When compared with general alert correlation approaches [12,13,15,33] that apply the concept of one correlation schema to aggregate related alerts, our alert aggregation module has a distinctive feature because we use different aggregators and their corresponding sub alert aggregators to process alerts from different IDS products. This means that our approach has an architecture that supports aggregation of alerts from multiple IDS products. Actually the act of handling a specific class of attack by a given alert aggregator and its corresponding sub aggregator for a given class of attack definitely simplifies the process of aggregating alerts, i.e. attacks from different classes are separated appropriately, hence contributing to better results. In addition, our aggregation module is fed with validated alerts as input and processes and outputs them in the form of aggregated meta alerts. Moreover, our alert aggregation module processes alerts of high quality (validated alerts), thus giving better results, unlike other similar correlation approaches [13,15,33] which are fed with unverified alerts. As noted earlier, the alert verification process filters out any alert without a corresponding vulnerability, thereby drastically reducing the overhead (processing time and storage) associated with unverified and unnecessary alerts. As a matter of fact, if an alert aggregation system receives unverified alerts that contain positives as input, then the quality of the aggregation results may degrade significantly such that the false positive alerts may be misinterpreted or, worse still, given undue attention.

In addition, when compared with other similar approaches, our approach not only aggregates alerts from multiple IDSs but also goes further to aggregate meta alerts. Our approach does not disregard alerts that are not shared across multiple IDS products, unlike probability-based approaches such as Valdes and Skinner [13] and Thomas *et al.* [5] that pay much attention to alerts that are shared by

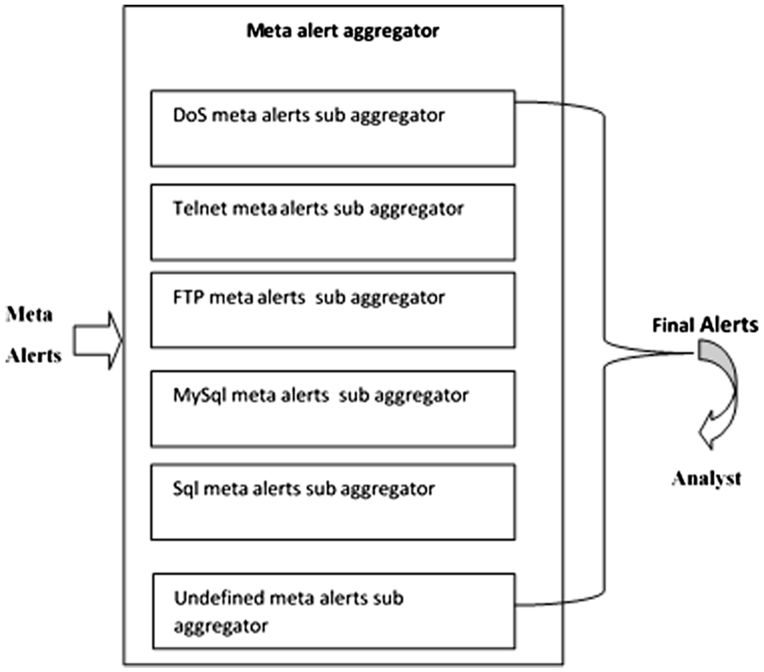
Meta alert aggregator
DoS meta alerts sub aggregator
Telnet meta alerts sub aggregator
FTP meta alerts sub aggregator
MySql meta alerts sub aggregator
Sql meta alerts sub aggregator
Undefined meta alerts sub aggregator
Meta Alerts
Final Alerts
Analyst

Figure 8. Aggregation of meta alerts

many IDS products and may ignore alerts from an individual IDS product (because they are not shared by other IDS products). We feel that the assumption applied by the probabilistic based approaches may affect the accuracy and detection rates of IDS products in general, hence influencing the final quality of alerts negatively. In fact, the main reason why the aforementioned approaches may not provide reliable results is because different IDS products may have different accuracy and detection rates as well as different speeds to detect the intrusions. Finally, unlike approaches such as Valeur [1], Valdes and Skinner [13] and Sourour *et al.* [16] that use fixed values of $\text{M.Time}_{\text{out}}$ and EC_{time} of a given intrusion to aggregate alerts, our alert aggregation module assigns different values for $\text{M.Time}_{\text{out}}$ and EC_{time} depending on the nature of the IDS product and the class of attack, as well in order to effectively aggregate alerts.

4. EXPERIMENT AND DISCUSSION

This section describes how we performed the experiment to validate the effectiveness of the proposed approach. As mentioned earlier, the proposed approach takes raw alerts generated by different IDS products as input, processes them using the alert verification and alert aggregation modules and delivers them as output in the form of meta alerts. To validate the proposed approach, a test bed was set up.

We used three different network signature-based IDSs, namely Snort [34], Prelude [35] and Shoki [36]. The IDS products were used with their default set of rules, since most signature-based IDSs run with their default rules. We chose one computer to provide storage services for alerts as well as housing the CVA data. The testing machines have the following specifications: Intel Core Duo processor (P8400) running at 2.56 MHz with 2 GB RAM. The testing machines are grouped into two: target machines and attacking machines. The target machines are further categorized into two:

1. Servers to provide services such as FTP, Telnet, SQL and MySQL services. Five servers were used (three Windows based, two Linux based).
2. Clients to receive services from servers. Fifteen clients were used (nine Windows based, six Linux based).

The target machines have different operating systems, such as Windows (Windows server 2003 and 2008, XP, Vista, 7) and Linux (Red Hat, Fedora, Ubuntu). In addition, application software was installed, such as Microsoft office (2000, 2003, 2007, 2010), Web browsers (Firefox, Internet Explorer, Google Chrome), OpenOffice, database management software (MySQL, SQL) and Adobe applications. In order to produce the desired results, we loaded the target machines with applications such as FTP server, Telnet, MySQL server and SQL server.

The attacking machines were run using different operating systems. Five attacking machines were used (three Windows based, 2 Linux based). Attacking machines were installed with widely known attacking tools such as Metasploit tool [37] in order to generate exploits and vulnerabilities of the target machines in the test bed. Two major types of attacks were executed: relevant attacks have the ability to exploit the vulnerabilities of the target machines. We further categorized the relevant attacks into two: (i) attacks that fully exploit vulnerabilities; and (ii) attacks that partially exploit vulnerabilities. The second major type of attacks are the non-relevant attacks. These attacks do not have the ability to exploit any vulnerability of the target machines. The attacks are categorized into five classes, namely DoS, FTP, SQL, MySql and Telnet.

This experiment used attacking machines to execute attacks on the target machines. We employed known techniques to exploit the vulnerabilities on different operating systems, software, protocols and ports. The attacks are executed in the test bed using the available attacking tools against the target machines. Examples of specific scenarios for multi-step attacks based on known techniques and exploits are: (i) a multi-step attack that exploits the vulnerabilities of the FTP server in the target machine. In this scenario, the attacking machine uses NMap to try to detect whether an FTP service is running on the target machine (FTP attack); (ii) a multi-step attack that involves attacking machines in order to interrupt the TCP service running in target machines. ICMP Flooder can help emulate the attack because it can continuously transfer large packets to the target machine, hence disrupting TCP service (DoS attack). Other specific attacks include: DoS attacks (Teardrop, Winnuke, Syndrop, etc.); FTP attacks (Finger redirect, Freeftpd username overflow, etc.); Telnet attacks (Telnet username buffer overflow, resolve host conf, etc.);SQL attacks (SQL server buffer overflow, SQL injection, etc.); MySQL (SSL hello message overflow, etc.).

We used two modes to deploy the IDS products: (i) each of the IDS products is deployed alone in the network in order to produce alerts; and (ii) different IDS products are employed in the network at the same time in order to produce comparable alerts. It is important to note that some attacks are run repeatedly for a considerable period of time. We implemented the alert verification and alert aggregation modules in JAVA. The architectures are shown in Figures 9 and 10.

4.1. CVA data

As mentioned in Section 3, CVA data comprise a comprehensive and in-depth database that represents the threat profile of our test bed. It is implemented in MySQL. CVA data list all the vulnerabilities by their reference i.d.s, names, severity, IP addresses, ports, protocol, class, time and applications. In a nutshell, the threat profile generator constructs CVA data by drawing data from four sources: scan reports produced by different vulnerability scanners such as Nessus [38] and NMap [39]; popular known vulnerability databases such as CVE; reference details from different IDS products (Snort, Prelude, Shoki); and finally the details of network resources of a given network. We used four sources to identify the relevant vulnerabilities representing the actual current threats of the network, as shown in Figure 5.

To generate reports of vulnerabilities, we employed scanners that are interoperable. The scanners search for missing patches, service packs, vulnerable software versions, vulnerable applications, services and protocols. The reason why we use multiple scanners is because they have different techniques to detect vulnerabilities. Scanners are configured to run periodically to extract the most consistent and recent threat view of the test bed. The reports are processed by Perl scripts. Special agents (Perl scripts) are installed in the target machines to track changes in applications, services and configuration. The necessary updates are reflected on the database of network resources, as are the CVA data.

As noted in Section 3, CVA data and IDS products may use different attack details such as reference i.d.s and names when referring to the same intrusion and therefore could have an impact when computing for alert relevance scores and the management of alerts in general. Standardization of attack and vulnerabilities details is a global issue and so far there are no unique attack details such as reference i.d.s to reference all types of attacks. Standardized schemes such as CVE have been developed to uniquely reference and directly map attack information to vulnerabilities. Snort, Prelude and Shoki refer their signatures to standard CVE. However, not all attacks have assigned CVE i.d.s as

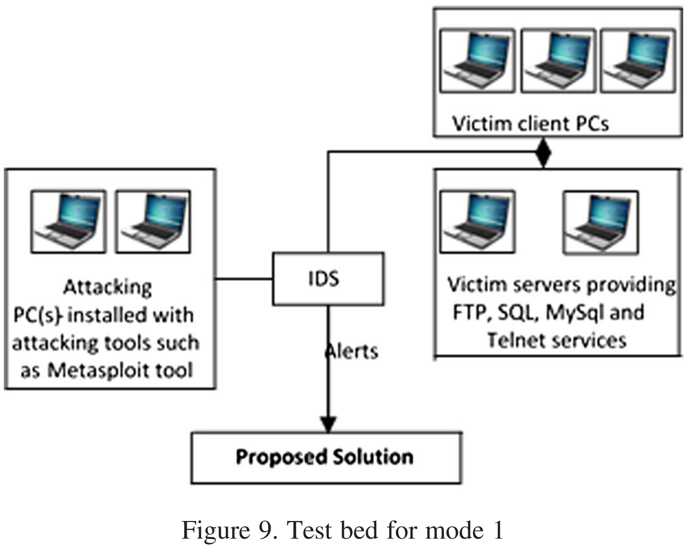
Victim client PCs
Attacking PC(s) installed with attacking tools such as Metasploit tool
IDS
Victim servers providing FTP, SQL, MySql and Telnet services
Alerts
Proposed Solution

Figure 9. Test bed for mode 1

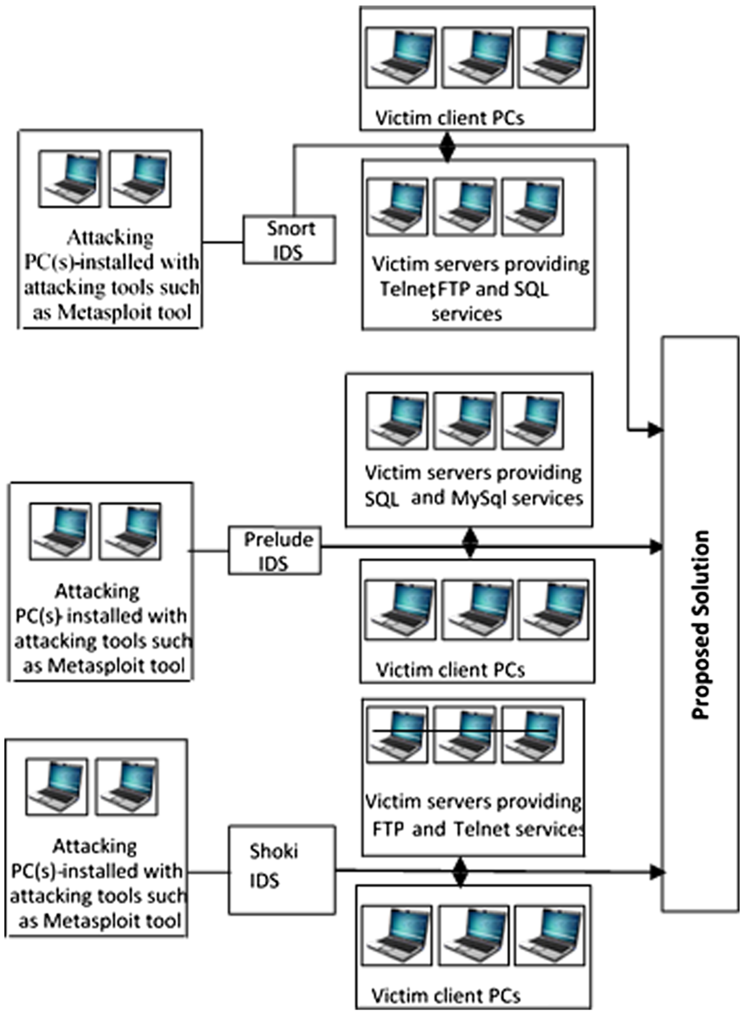
Victim client PCs
Attacking PC(s)-installed with attacking tools such as Metasploit tool
Snort IDS
Victim servers providing Telnet,FTP and SQL services
Victim servers providing SQL and MySql services
Prelude IDS
Attacking PC(s)-installed with attacking tools such as Metasploit tool
Victim client PCs
Victim servers providing FTP and Telnet services
Attacking PC(s)-installed with attacking tools such as Metasploit tool
Shoki IDS
Victim client PCs
Proposed Solution

Figure 10. Test bed for mode 2

reference i.d.s. For example, 33–42% of attack signatures in Snort (since 2003) have CVE i.d.s [16,19]. Missing CVE details could have an impact on alert validation. The most appropriate solution is to establish a mechanism to map identifiers for attacks and vulnerabilities. In our bid to resolve this issue, we examined attack details such as reference i.d.s and names of vulnerabilities and attacks contained in CVA data and multiple IDS products. This calls for the need to determine the coverage overlap of standardized attack details between CVA data and multiple IDS products and then to include additional attack details based on different IDS products in the CVA data.

We have proposed a mapping facility to help in setting up additional mappings based on Snort-, Prelude- and Shoki-specific identifiers into CVA data (refer to Table 2). This involves the following:

- Examining the attack reference details contained in the CVA data and the three IDS products.
- Determining the coverage overlap of standardized attack reference details between CVA data and Snort, Prelude and Shoki.
- Setting up additional attack reference details based on three IDS products into CVA data with the help of the OSVDB, which is an open source database for vulnerabilities that references CVE, Snort, Prelude, Shoki, Nessus and other leading network security products. That is, the CVA data use other IDS products' references such as Bugtraq i.d.s and OSVDB i.d.s when CVE i.d.s are not applicable (complement CVE when referring to attacks).

We limited the coverage of this exercise to the attacks used in the test bed. We added the attack information into the CVA data to ensure that the details derived were appropriate and would positively influence the alert validation and computation of the alert relevance score as well. This exercise brought the benefit of completeness of attack details in the CVA data because, if attack details are missed, this affects the alert relevance score negatively.

4.2. *Performance of the proposed approach*

Here we focus on the performance of different components of the proposed approach. When constructing CVA data, we noted that the time taken to build these data was determined by factors such as the number of IDS products deployed in the test bed, the number of target machines and applications hosted in each target machine, and the type and number of vulnerability scanners in the test bed. The average time to scan one target machine is 2–6 minutes depending on the above-mentioned factors. We took 60 minutes to test each class of attack.

Our experiment generated 931 Snort alerts, 912 Prelude alerts and 965 Shoki alerts, as shown in Table 4. The alerts shown in Table 4 conform to the number of attacks executed in our experiment. That is, about 15% of alerts represent attacks that exploited the vulnerabilities (relevant attack) of the test bed, while 85% of alerts represent attacks that did not exploit the vulnerability (non-relevant attack).

We collected and pre-processed alerts using Perl scripts, as discussed in Section 3. The pre-processed alerts were fed as input into the verification module for validation and then forwarded to the alert aggregation module in order to reduce redundant and isolated alerts. We configured the alert aggregation component with the following parameters:

- $\text{M.Time}_{\text{out}} = 1\text{–}3$ minutes;
- $\text{EC}_{\text{time}} = 2\text{–}14$ minutes (depending on classes of attacks).

The performance of the proposed approach is presented in two categories: (i) performance evaluation of alert verification module; and (ii) performance evaluation of alert aggregation module.

4.2.1. *Performance evaluation of alert verification module*

We employed two metrics in order to evaluate the effectiveness of the alert verification module:

a) Detection rate: indicates the number of attacks that are detected by IDS among all relevant attacks that are generated. We applied the following equation: detection rate = TP/(TP + FN).
b) Accuracy: indicates the percentage of relevant attacks detected versus all cases detected as attacks by IDS. We applied the following equation: accuracy = TP/(TP + FP).

where TP (true positive) refers to attacks that are correctly detected; FP (false positive) refers to attacks that are incorrectly detected; and FN (false negative) refers to attacks that are not detected.

We used the above metrics to measure the effectiveness of the alert verification module before and after alert validation, as shown in Tables 4 and 5. The results presented in Table 4 reveal that the majority of the alerts are due to ineffectiveness of the different IDS products. Table 4 further reveals that different IDS products detect most of the relevant attacks. However, IDS products perform poorly on non-relevant attacks and hence generate many false positives.

In addition, we observed that different IDS products may generate different outputs (alerts) for a given attack. Some IDS products may perform well on specific attacks. This is further illustrated by difference in accuracy and detection rates, as shown in Table 4. For example, the accuracy of alerts generated by Snort IDS is between 12% and 14%. Actually, in the literature, Snort has an average accuracy of about 12%. Snort records a slightly higher accuracy than other IDS products because Snort has a relatively comprehensive signature database. Similarly, other IDS products (Prelude and Shoki) recorded a lower accuracy, as shown in Table 4. The low accuracy manifested in raw alerts across different IDS products justifies the need to employ an alert verification technique to further improve the accuracy of alerts. Table 4 continues to reveal that Snort IDS exhibited a higher detection rate of between 90% and 100% when compared with other IDS products. To further explore the impact of the alert verification module, we describe Table 4 in depth:

Table 4. Accuracy and detection rate of raw alerts (before alert verification)

	Snort					Prelude					Shoki				
	DoS	Telnet	FTP	MySQL	SQL	DoS	Telnet	FTP	MySQL	SQL	DoS	Telnet	FTP	MySQL	SQL
Attacks	92	143	399	196	121	89	145	389	192	119	90	146	394	191	117
Relevant attacks	14	20	57	28	18	12	21	61	24	17	13	22	55	26	16
Non-relevant attacks	78	123	342	168	103	77	124	328	168	102	77	124	339	165	101
Raw alerts collected (unverified)	89	141	386	189	126	85	140	379	185	123	80	159	407	197	122
TP	13	18	56	27	18	10	19	59	21	15	10	19	53	24	13
FP	75	121	329	161	108	73	119	318	161	106	67	137	352	171	106
FN	1	2	1	1	0	2	2	2	3	2	3	3	2	2	3
Accuracy (%)	14.7	12.9	14.5	14.3	14.2	12.0	13.7	15.6	11.5	12.3	12.9	12.1	13.0	12.3	10.9
Detection rate (%)	92.8	90.0	98.2	96.4	100	83.3	90.4	96.7	87.5	88.2	76.9	86.3	96.3	92.3	81.2

Table 5. Accuracy and detection rates of transformed alerts (alert verification module)

		DoS	Telnet	FTP	MySQL	SQL
Accuracy rates	Before alert verification	13.2	12.9	14.36	12.7	12.46
	After alert verification	87.43	96.46	97.03	94.33	92.96
Detection rates	Before alert verification	84.33	88.9	97.06	92.06	89.8
	After alert verification	74.06	85.83	94.23	86.9	83.8

- First row: indicates the number of attacks generated in our test bed.
- Second row: indicates the number of relevant attacks generated in our test bed.
- Third row: indicates the number of non-relevant attacks generated in our test bed.
- Fourth row: indicates the raw alerts from different IDS products.
- Fifth row: indicates true positive (TP) alerts.
- Sixth row: indicates false positive (FP) alerts.
- Seventh row: indicates false negative (FN) alerts.
- Eighth row: indicates accuracy rate before alert verification.
- Ninth row: indicates detection rate before alert verification.

Table 5 displays the results after employing the alert verification module in terms of accuracy and detection rates for different classes of attacks. As demonstrated in Table 5, the proposed scheme has improved the accuracy of alerts. The alert verification module filters out alerts that show little or no similarity when compared to the threat profile of the test bed. As seen in Table 5, the alert verification module improves the accuracy of alerts generated by different IDS products. The table reveals that the alert verification module minimally affected the detection rates of IDS products after the alert verification process. Several factors may have contributed to lower detection rates. The alert verification module may eliminate some alerts with cases involving unknown vulnerability or attack, hence lowering the detection rate. Consider the following scenario: an IDS product may issue an alert for a successful attack but CVA data have not registered a corresponding vulnerability for this attack and therefore the relevant alerts may be ignored or considered as false alerts because they fail to match the vulnerabilities in CVA data during the alert verification process. To minimize the influence of such scenarios in alert management, we suggest that IDS products and CVA data should be updated frequently in order to address the issue of unknown vulnerability and attacks. Finally, we note that the alert verification module is effective in eliminating non-relevant alerts generated by different IDS products and is able to successfully separate false alerts from true alerts.

Figures 11 and 12 show bar graphs that compare the accuracy and detection rates of alerts before and after processing them by the proposed system. Figure 11 shows the accuracy rates of alerts before and after alert verification. The proposed scheme has greatly improved the accuracy of the raw alerts due to a robust alert verification process incorporated in this scheme. For example, our scheme improved the accuracy rates of raw alerts reporting DoS attacks from 13.2% to 87.43%, as illustrated in Figure 11. Similarly, Figure 12 shows the detection rates of raw alerts as well for validated alerts. As noted earlier, the proposed scheme has minimally affected the detection rates of raw alerts.

4.2.2. *Performance evaluation of alert aggregation module*

As noted earlier, the alert verification does not guarantee final alerts of good quality since the validated alerts may be redundant and isolated alerts. Redundant alerts are usually generated by multi-step attacks and may overwhelm the analysts and therefore present challenges when analyzing the attack trends. During this experiment, we observed that IDS products generate huge volumes of alerts within a short period when detecting multi-step intrusions. This concept is further illustrated when IDS products generated an overwhelming number of alerts in the first 2 minutes of the test period when detecting Ping of Death attack. Further, the IDS products failed to establish the relationships among these alerts. The main goal of the alert aggregation module is to reduce the huge number of redundant and isolated alerts after the alert verification process.

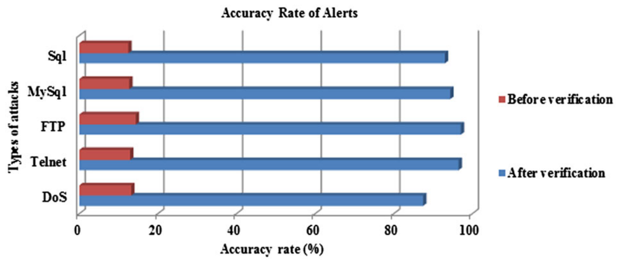

Figure 11. Accuracy rates

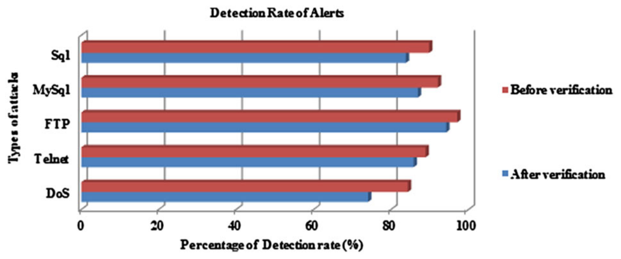

Figure 12. Detection rates

We evaluated the effectiveness of the alert aggregation module by employing the alert reduction rate metric. This metric shows the percentage of filtered alerts (eliminated) by the aggregation module. We used the following equation:

$$\text{reduction rate} = \text{filtered alerts}/\text{total number of alerts}$$

The results of alert aggregation are presented in Table 6. The table displays the number of non-aggregated alerts (validated alerts) and the aggregated alerts (meta alerts) in each class of attack. The table further shows the reduction rates after employing the alert aggregation module.

As seen in Table 6, the module reduces validated alerts significantly by rates of between 44.4% and 59.5%. It is crucial to note that the these reduction rates do not take into account several alerts that are filtered (eliminated) by the alert verification module. We observed that the alert sub verifiers forward the validated alerts to their corresponding sub alert aggregator correctly. This is explained by the fact that the aggregation module runs the appropriate aggregation algorithm for each sub aggregator.

In addition, we used an undefined attack sub aggregator to process any alert reporting a new attack (previously not known) in order to improve the management of alerts, thus increasing the detection rate of unknown attacks. We used the result of the undefined attack sub aggregator to improve the alert verification by streamlining the CVA data as well as the alert aggregation process.

Unlike existing alert correlation approaches, the proposed alert aggregation module further reduces redundant and isolated alerts from multiple IDS products. As noted earlier, aggregating alerts according to their sources (IDS products) may not be an effective way to reduce unnecessary alerts. In fact, when a given attack occurs in a network, different IDS products may respond by generating alerts that have similar features such as source address, destination address and reference i.d.s. Such alerts need to be aggregated together based on their common features (such as reference i.d.s, source and destination addresses) in order to further reduce redundant alerts.

As illustrated in Table 7, the alert aggregation module further reduces redundant and isolated alerts across the IDS products. The table shows two types of meta alerts: shared and non-shared meta alerts. Shared meta alerts refer to meta alerts that are shared by a given set of IDS products (in terms of similar features such as source and destination addresses). For example, the first row of the column named ‘IDS product’ shows that there are 78 meta alerts that are shared across the three IDS products. The alert aggregation module further reduces the 78 meta alerts to 22 final meta alerts (by a rate of 71.7%). From this table we can deduce that our alert aggregation module has the ability to further

Table 6. Alert reduction on validated alerts

Sub aggregator	Number of validated alerts	Remaining alerts after using alert aggregation module (in the form of meta alerts)	Reduction rate (%)
DoS	33 (14 ideal, 19 partial)	17 (6 ideal, 11partial)	48.48
Telnet	56 (23 ideal, 33 partial)	27 (9 ideal, 18 partial)	51.78
FTP	168 (60 ideal, 108 partial)	68 (17 ideal, 51 partial)	59.5
MySQL	72 (29 ideal, 43 partial)	40 (13 ideal, 27 partial)	44.4
SQL	46 (18 ideal, 28 partial)	24 (8 ideal, 16 partial)	47.8

reduce the shared meta alerts (the meta alerts exhibit similarity of features such as source and destination addresses) generated by a given set of IDS products. The second type of meta alert, the non-shared meta alerts, refer to meta alerts that are not shared by a given set of IDS products.

It is worth noting that the alert aggregation module successfully reduces redundant and isolated alerts of shared meta alerts by a reduction rate of 68.4–71.7%, as illustrated in Table 7. Finally, we reveal that it is very difficult to know the attack patterns if the attack period exceeds the expected exploit cycle time since our module only targets meta alerts generated within the expected exploit cycle time.

4.3. *Processing time*

We measured the maximum time taken to process alerts in two modules (alert verification and aggregation modules). We noted that the alert verification module takes 0.031 seconds to successfully complete the process of validating an alert. In addition, we noted that an alert aggregation module takes 0.194 seconds to aggregate similar alerts (including time taken to aggregate similar alerts generated by different IDS products). Therefore the proposed approach takes a maximum of 0.225 seconds to successfully process an alert in the two modules. Finally, we note that the proposed system presents quality alerts and does not introduce unnecessary delays; hence analysts are able to react early to different intrusions.

4.4. *Overall evaluation*

As noted earlier the proposed system has two modules that improve the quality of alerts. In a nutshell, the alert verification module filters out alerts with no corresponding vulnerability and the alert aggregation module reduces the redundant and isolated alerts after alert verification. The system is unique considering the fact that we are dealing with alerts from multiple IDS products. We remark that the two modules of the proposed approach can be implemented in a network with multiple signature-based IDS products without configuring the default signatures. In fact, CVA data require to be updated with attack reference details of those IDS products accordingly in order to have comprehensive network-specific vulnerabilities.

Table 8 compares the performance between the proposed approach and other similar approaches. The proposed approach recorded the highest alert reduction rates in terms of false positives and

Table 7. Further meta alert reduction

Meta alerts	IDS product	Initial meta alerts (input)	Final meta alerts (output)	Reduction rate (%)
Shared meta alerts	Snort, Prelude, Shoki	78	22	71.7
	Snort and Prelude,	19	6	68.4
	Snort and Shoki	16	5	68.7
	Prelude and Shoki	13	4	69.2
Non-shared meta alerts	Snort	13	13	0.0
	Prelude	18	18	0.0
	Shoki	19	19	0.0

redundant positive alerts. Our scheme recorded superior performance in eliminating the false positives due to its robust alert verification engine. The scheme further reduces redundant validated alerts in the form of meta alerts. The scheme is able to efficiently reduce huge volumes of unnecessary alerts generated by multiple IDS products, unlike other existing approaches.

In terms of scalability, the proposed approach fared well. The scheme has a very robust algorithm that is able to efficiently reduce a huge number of unnecessary alerts generated by multiple IDS products. The proposed scheme has the highest number of raw alerts (2808) as input alerts and recorded the highest alert reduction rate, as mentioned earlier (shown in Table 8). However, the scheme performed better in terms of accuracy and detection rates because of the nature of alerts being processed. The scheme is designed to handle alerts generated from multiple IDS products, unlike the other two approaches. It is a fact that different IDS products have different accuracy and detection rates and therefore have a negative impact on our scheme. It is important to note that one must take into account that some IDS products have better detection and accuracy rates than others. Some IDS products might generate alerts more slowly than others; hence this must be taken into consideration when aggregating the alerts. In order to make the scheme more scalable in terms of alert reduction, accuracy and detection rates when handling the overwhelming number of alerts from multiple IDS products in large networks, we plan to explore the development of a more efficient mapping facility to map attack reference i.d.s into CVA data.

In a nutshell, the proposed approach presents numerous benefits as far as alert management is concerned. For example, we recorded superior performance in reducing huge volumes of unnecessary alerts from multiple IDS products, as noted earlier. In addition, our system does not require extensive training.

5. CONCLUSION

We have presented an approach to manage unnecessary alerts generated by multiple IDS products. The critical components of the approach are alert verification and alert aggregation modules. The alert verification module uses CVA data to improve the accuracy and quality of the raw alerts from multiple IDS products by filtering out those with no corresponding vulnerability in a given network. The alert aggregation module reduces redundant and isolated alerts generated by different IDS products. We carried out an experiment that involved processing alerts from several classes of attacks using three different IDS products and recorded high accuracy and reduction alerts of the IDS products.

We encountered several limitations that need to be addressed in order to obtain optimal results. Some of the limitations include inefficient correlation between alerts and corresponding vulnerabilities because of how IDS products and vulnerabilities reference each other. Further efforts should focus on creating a more scalable and flexible automated mapping facility to map attack reference i.d.s into CVA data, as noted earlier. There is a need to focus on mechanisms and policies in order to have an effective solution towards reducing the influence of unknown attacks and vulnerabilities on alert management. Moreover, our future work will include anomaly-based IDS in order to offer a more comprehensive solution. We believe the

Table 8. Performance comparison

Approach	Sourour *et al.* [16]	Njogu *et al.* [24]	Proposed scheme
Technique	Alert correlation	Vulnerability-based alert reduction for single IDS product	Vulnerability-based alert reduction for multiple IDS products
Initial alerts (input)	753	904	2808
Reduction rate of false positives (%)	30.9	78.2	86.6
Reduction rate of redundant positive alerts (%)	33.0	52.4	76.8 (reduced redundant validated alerts and redundant meta alerts)
Accuracy rate (%)	10.9	96.1	93.6
Detection Rate (%)	95.3	92.6	84.9

anomaly-based IDS can detect new attacks that are not detected by signature-based IDSs. We will focus on how we can aggregate alerts from signature-based IDSs and anomaly-based IDSs. In order to further evaluate the performance of the proposed approach, we plan to conduct more experiments with real traffic tracing which will include alerts generated by multiple IDS products. Lastly, part of our future work is to integrate AI techniques and algorithms, particularly those that handle the representation of logic as well as reduction in the size of the database (alerts) in order to improve the alert management process, hence improving the alert verification and aggregation processes and reducing unnecessary alerts.

ACKNOWLEDGEMENT

This work is supported by Hunan Provincial Natural Science Foundation of China (Grant No. 13JJ2017).

REFERENCES

1. Valeur F, Vigna G, Kruegel C, Kemmerer RA. A comprehensive approach to intrusion to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing* 2004; **1**(3): 146–169.
2. Tjhai GC, Furnell SM, Papadaki M, Clarke NL. A preliminary two-stage alarm correlation and filtering system using SOM neural network and K-means algorithm. *Computers and Security* 2010; **29**: 712–723.
3. Hubballi N, Biswas S, Nandi S. Network specific false alarm reduction in intrusion detection system. *Journal of Security and Communication Networks* 2011; **4**(11): 1339–1349.
4. Pietraszek T, Tanner A. Data mining and machine learning: towards reducing false positives in intrusion detection. *Information Security Technical Report 2005* 2005; **10**(3): 169–183.
5. Thomas C, Balakrishnan N. Improvement in intrusion detection with advances in sensor fusion. *IEEE Transactions on Information Forensics and Security* 2009; **4**(3): 542–550.
6. Gula R. Correlating IDS alerts with vulnerability information, revision 4. Technical Report, Tenable Network Security: Columbia, MD, 2011.
7. Kruegel C, Roberstson W, Vigna G. Using alert verification to identify successful intrusion attempts. *Praxis der Informationsverarbeitung und Kommunikation* 2004; **27**(4): 219–227.
8. Chaboya DJ, Raines RA, Baldwin RO, Mullins BE. Network intrusion detection: automated and manual methods prone to attack and evasion. *IEEE Security and Privacy* 2006; **4**(6): 36–43.
9. Michele C, Daniele G, Mirco M. Selective alerts for the run-time protection of distributed systems. In *Proceedings of the Ninth International Conference on Data Mining, Protection, Detection and other Security Technologies (DATAMINING 2008).*
10. Morin B, Mé L, Debar H, Ducassé M. M2d2: a formal data model for IDS alert correlation, In Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID). Springer: Berlin, 2002; 115–137.
11. Al-Mamory SO, Zhang H. A survey on IDS alerts processing techniques, In *Sixth WSEAS International Conference on Information Security and Privacy*, Tenerife, Spain, 14–16 December 2007; 69–77.
12. Julisch K. Clustering intrusion detection alerts to support root cause analysis. *ACM Transactions on Information and System Security* 2003; **6**(4): 443–471.
13. Valdes A, Skinner K. Probabilistic alert correlation. In Fourth International Symposium on Recent Advances in Intrusion Detection RAID. Lecture Notes in Computer Science 3224. Springer: Berlin, 2001; 54–68.
14. Debar H, Wespi A. Aggregation and correlation of intrusion-detection alerts. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2001; 85–103.
15. Al-Mamory SO, Zhang H. Intrusion detection alerts reduction using root cause analysis and clustering. *Computer Communications* 2009; **32**: 419–430.
16. Sourour M, Adel B, Tarek A. Network security alerts management architecture for signature-based intrusion detection systems with a NAT environment. *Journal of Network System Management* 2011; **19**: 472–495.
17. Jan N, Lin S, Tseng S, Lin N. A decision support system for constructing an alert classification model. *Expert Systems with Applications* 2009; **36**: 11145–11155.
18. Ning P, Reeves S, Cui Y. Correlating alerts using pre-requisites of intrusions. Technical Report TR-2001-13, North Carolina State University, 2001.
19. Massicotte F, Couture M, Labiche Y. Context based intrusion detection using Snort, Nessus and Bugtraq databases. In *Proceedings of the Annual Conference on Privacy, Security and Trust*, 2005; 1–12.
20. Porras PA, Fong MW, Valdes A. A mission impact based approach to INFOSEC alarm correlation. In Fifth International Symposium on Recent Advances in Intrusion Detection. Springer: Berlin, 2002; 95–114.
21. Morin B, Me L Debar H, Ducssse M. A logic-based model to support alert correlation in intrusion detection. *Information Fusion* 2009; **10**: 285–299.
22. Bakar NA, Belaton B. Towards implementing intrusion detection alert quality framework. In *Proceedings of the First International Conference on Distributed Framework for Multimedia Applications*, 2005; 196–205.
23. Njogu HW, Jiawei L. Using alert cluster to reduce IDS alerts. In *Third IEEE International Conference on Computer Science and Information Technology*, 2010; 467–471.

24. Njogu HW, Jiawei L, Kiere JN. Network specific vulnerability based alert reduction approach. *Journal of Security and Communication Networks* 2013; **6**(1): 15–27.
25. Yu J, Reddy YVR, Selliah S, Kankanahalli S, Reddy S, Bharadwaj V. TRINETR: an architecture for collaborative intrusion detection and knowledge-based alert evaluation. *Advanced Engineering Informatics* 2005; **19**: 93–101.
26. Liu X, Xiao D, Peng X. Towards a collaborative and systematic approach to alert verification. *Journal of Software* 2008; **3**(9): 77–84.
27. Michele C, Gozzi D, Marcjetti M. Selective and early threat detection in large networked systems, In *Proceedings of the Tenth IEEE International Conference on Computer and Information Technology (CIT 2010)*, Bradford, UK, June 2010.
28. Bolzoni D, Crispo B, Etalle S. ATLANTIDES: an architecture for alert verification in network intrusion detection systems. In *21st Large Installation System Administration Conference, LISA 2007*, Dallas, TX, 11–16 November 2007.
29. IDMEF. *RFC 4765*. Available: www.ietf.org/rfc/rfc4765.txt [7 March 2014].
30. Common Vulnerability and Exposures (CVE). Available: http://www.cve.mitre.org/about [7 March 2014].
31. OSVDB. Available: http://www.osvdb.org/ [7 March 2014]
32. Browne HK, Arbaugh WA, McHugh J, Fithen WL. A trend analysis of exploitations. In *Proceedings of IEEE Symposium on Security and Privacy*, 2001; 214–229.
33. Lee S, Chung B, Kim H, Lee Y, Park C, Yoon H. Real-time analysis of intrusion detection alerts via correlation. *Computer and Security* 2006; **25**: 169–183.
34. Snort. Available: www.snort.org [7 March 2014].
35. Prelude. Available: https://www.prelude-ids.org/ [7 March 2014]
36. Shoki. Available: http://sourceforge.net/projects/shoki/files/ [7 March 2014].
37. Metaspoilt. Available: www.metasploit.com [7 March 2014].
38. Nessus. Available: www.nessus.org [7 March 2014].
39. NMap. Available: www.nmap.org [7 March 2014].

AUTHORS' BIOGRAPHIES

Tu Hoang Nguyen received his M.Sc in Computer Science from College of Information Science and Engineering, Hunan University, Changsha, Republic of China. After completing his Masters program, he got the distinguished student scholarship to for a PhD program in Computer Science at Hunan University. His research interest include Intrusion detection and prevention, vulnerability analysis, network security, bioinformatics, and data mining.

Jiawei Luo is a full professor and vice dean at the College of Information Science and Engineering, Hunan University, Changsha, Republic of China. She holds Ph.D, M.Sc. and B.Sc. degrees in Computer Science. Her research interests include data mining, network security and bio informatics. She has a vast experience in implementing national projects on Bioinformatics. She has authored many research articles in leading international journals.

Humphrey Waita Njogu is currently an IT Security Expert working in a government agency in Kenya. He received his Ph.D and M.Sc in Computer Science (Security) from Hunan University, China. He received his B.Sc. degrees in Information Science from Moi University, Kenya. He holds several IT professional certifications in Cisco, Oracle, Comptia, Linux and Microsoft. His research interests include most aspects of IT security, with an emphasis on network security, Intrusion detection and prevention, vulnerability analysis, data mining, Green Computing, Cloud computing security and Social media security. He has a vast experience in implementing several IT security projects. He has authored many IT security research articles in leading ISI/SCI top indexed international journals and conferences.