Improving Intrusion Detection through Alert Verification

Thomas Heyman, Bart De Win, Christophe Huygens and Wouter Joosen

IBBT / DistriNet
Katholieke Universiteit Leuven, dept. of Computer Science
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

Abstract. Intrusion detection systems (IDS) suffer from a lack of scalability. Alert correlation has been introduced to address this challenge and is generally considered to be the major part of the solution. One of the steps in the correlation process is the verification of alerts. We have identified the relationships and interactions between correlation and verification. An overview of verification tests proposed in literature is presented and refined. Our contribution is to integrate these tests in an extensible generic framework for verification that enables further experimentation. A proof-of-concept implementation is presented and a first evaluation is made. We conclude that verification is a viable extension to the intrusion detection process. Its effectiveness is highly dependent on contextual information.

1 Introduction

The importance of computer security is increasing. Besides implementing security measures at the application level, operating system level and network level, there is also a need for monitoring the protected infrastructure after deployment. This is what intrusion detection systems (IDS) attempt to do: by monitoring local audit trails on a computer, for example, host-based IDS attempt to detect local attacks. Network-based IDS, on the other hand, attempt to detect attacks on the network they are monitoring.

In general, an IDS observes events and tries to detect signs of attacks. These attacks, when successful, result in an intrusion. A correctly identified attack generates an alert, called a *true positive*. Conversely, the absence of an alert in situations where no attacks are executed is a *true negative*. IDS can make mistakes: an IDS can generate an alert when no attack actually took place, this is a *false positive*. False positives are generated because of insufficiently strict defined rules in a misuse- or policy-based IDS, or an insufficiently trained anomaly-based IDS, among others. The absence of an alert when an attack did happen is a *false negative*. A *non-relevant positive* (or non-contextual positive, [1]) is an alert generated because of an actual attack, but this attack could never result in an intrusion, for example because the target is not vulnerable.

Intrusion detection systems are faced with *four basic problems* (as stated in [2]) that need to be solved in order to make intrusion detection more scalable:

Heyman T., De Win B., Huygens C. and Joosen W. (2006).
Improving Intrusion Detection through Alert Verification.
In *Proceedings of the 4th International Workshop on Security in Information Systems*, pages 207-216

Too many alerts. IDS generate overwhelming amounts of alerts. Even in the case that all alerts are correct (i.e. true positives), a deluge of low-level alerts severely decreases the usability of the IDS for the operator.

False and non-relevant positives. The alert stream contains a large fraction of false alerts, both false and non-relevant positives.

Insufficient diagnostic information. The generated alerts are at a semantically low level: they do not provide sufficient information to the administrator to allow him to efficiently remedy the situation.

Low detection rate. The fourth concern is the detection rate, or false negative rate. IDS still miss a fraction of the actual number of attacks, failing to generate alerts.

These problems, when combined, make it very easy to lose overview of what is happening on the monitored infrastructure. While it is hard enough to try and get a comprehensive overview of the current attacks and intrusions from a large amount of low-level alerts containing lots of false and non-relevant positives, the presence of false negatives further complicate matters. For instance, some key attacks necessary to comprehend the complete attack strategy of the attacker might have been missed by the IDS.

Another issue is the amount of IDS sensors. Using a single IDS quickly becomes impossible for larger infrastructures. One host-based IDS reaches its limit when more than one system needs to be protected, the visibility of a network-based IDS is limited to the local network. However, simply deploying more IDS is no scalable solution, as this generates an over-abundance of alerts. The research community has developed a category of methods to counter these scalability problems, i.e. alert correlation.

Alert correlation has some problems of its own. False and non-relevant positives have a negative impact on correlation algorithms, as they might give rise to the generation of non-existent attack scenario's [1]. This is what alert verification tries to achieve: it helps to improve the quality of alerts passed on to the later correlation steps. While verification is not a complete solution by itself, it is an indispensable element of correlation that has not been studied in full depth.

We have studied the interaction between correlation and verification, as well as available methods proposed to verify alerts. The contribution of this paper lies in the integration of verification approaches in a generic, extensible framework for alert verification that enables experimentation. By using different heuristics, the relevance of an alert (i.e. is the alert a true positive) is verified. A proof-of-concept implementation of the verification framework is presented and evaluated. From our preliminary results, we conclude that verification is indeed a viable extension to the intrusion detection process.

The rest of this paper is structured as follows. First, a generic architecture for intrusion detection is presented in Section 2, introducing the processes of alert correlation and verification. We then present a framework for alert verification in Section 3 and a proof-of-concept implementation. We discuss the status of our and related work in Section 4 and we conclude in Section 5.

2 General Alert Correlation Architecture

In the first part of this section, a brief overview of an intrusion detection architecture is presented. The correlation and verification components are situated in this architecture. In the second part of this section, the use of verification to counter the scalability issues of intrusion detection is elaborated upon.

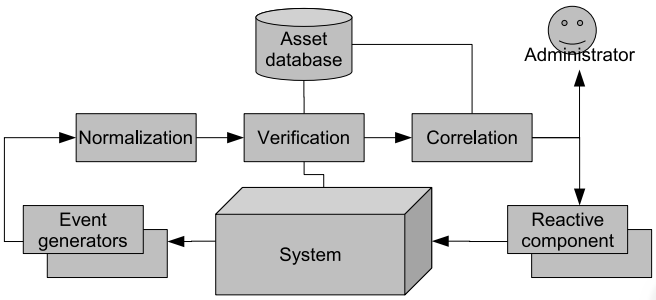

Fig. 1. Generic intrusion detection architecture, supporting verification and correlation.

A general intrusion detection architecture with verification and correlation components, is shown in Fig. 1. This architecture is loosely based on that presented in [1]. Events from the system being monitored are observed by different *event generators*. The generators, low-level IDS, process these events looking for certain attack signatures (in the case of misuse-based IDS) or anomalous behavior (in the case of anomaly-based IDS). If attacks are detected, alerts are generated and passed to the normalization component.

The *normalization component* accepts different alert formats from the heterogeneous sensors and outputs alerts in a normalized format — for example, the Intrusion Detection Message Exchange Format, IDMEF [3], which is an XML-based format to represent intrusion detection alerts. These normalized alerts are passed to a *verification engine*. This component tries to verify the alerts by distinguishing true positives from false and non-relevant ones, based on information from an asset database and probing of the system being monitored.

The *asset database* contains information on the the system being monitored. This information includes: the hosts being protected, services offered, vulnerability information, etc. It provides this information to the verification component and the correlation component. Asset data may be manually entered into the database, or gathered automatically through various processes such as network and vulnerability scans. This data may also be updated by the verification component as a side effect of the verification of alerts.

The verified alerts are then sent to the actual correlation engine. Alert correlation is the process of grouping and conceptually reinterpreting intrusion detection alerts. It accepts (low level) intrusion detection alerts, generates groups out of these alerts and

can assign new meanings to these groups. The output is a lower number of alerts on a higher semantic level (also called *meta-alerts*, [4, 1] e.a.). For example: independent detections of strange network packets can be grouped and reinterpreted as one port scan. A brief overview of correlation strategies is presented in Section 4.

The high-level alerts from the correlation component are then presented to the administrator. Or, in the case of intrusion prevention systems that are able to actively react to intrusions, the alerts can be fed to reactive components which then automatically attempt to prevent the attack from resulting in an intrusion.

The correlation process, by presenting higher-level meta-alerts to the administrator in stead of low-level sensor alerts, reduces the amount of alerts and is able to complement them with diagnostic information. Some correlation methods are even able to compensate for false negatives, by reasoning about attacks that might have been missed by the IDS [5]. However, as mentioned, false and non-relevant positives still have a negative impact on the correlation results.

Verification, as an intelligent pre-processing filter in the alert correlation process, separates false and non-relevant alerts from true positives, so that the actual correlation algorithms are provided with true positives only (in the ideal case). By enhancing the quality of the alert stream, it is a useful addition to every IDS: it helps to handle the first two IDS scalability issues and improves correlation results. Alert verification is not able to solve the other scalability problems, however. First of all, verification does not change the conceptual level of alerts. The low-level input alerts remain at their low semantic level. Verification is also unable to take false negatives into account. These issues are handled by the correlation process.

Individual sensors are generally insufficiently aware of the context to decide if an attack is likely to be real and could have resulted in an intrusion. The verification process, using information from the asset database or by actively probing systems or the network, is not, and can make this distinction. In principle, even more contextual information, like the security policy of a company, could be incorporated to distinguish between attacks that the company does not deem important (i.e. port scanning an outside firewall) and attacks that are. In this context, verification can be used as a partial replacement for a priori policy tuning of IDS[1].

Verification is able to handle intentionally generated false alerts, generated on purpose by an attacker.[2] While all false alerts (both intentionally and unintentionally generated false positives, non-relevant positives) are detrimental to IDS, intentional alerts should be especially avoided in intrusion prevention systems, as these systems are able to actively try to prevent the attack from resulting in an intrusion. The prevention functionality could be misused by an attacker to turn the IDS against legitimate users.

[1] A priori policy tuning is the process of adjusting the configuration of an IDS, before deployment, to the security policy of an organization.

[2] For example, if alerts are generated for port scans, an attacker could misuse nmap (http://www.insecure.org/nmap/), a popular port scanner, to fake the originating address or generate decoys ($-S$ and $-D$ options, respectively).

3 Verification Framework

3.1 Overview

In this section we present a framework for alert verification: it accepts normalized IDMEF alerts, executes various verification tests and outputs the same alerts with an added plausibility p. This is a floating point value ($p \in [0, 1]$) that expresses the level of confidence in the alert: $p = 0$ is an indication that the alert is a non-relevant or false positive, $p = 1$ suggests a true positive.

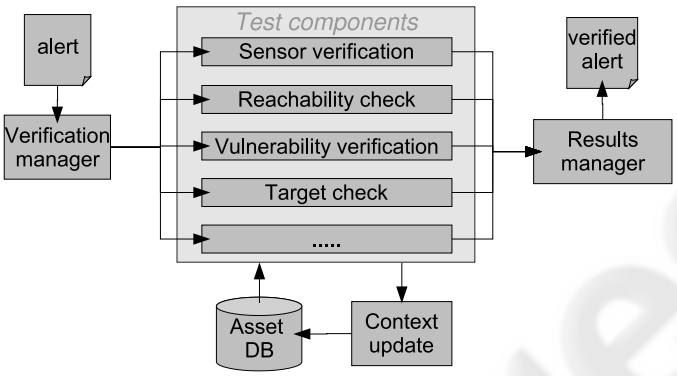

Fig. 2. The alert verification framework and implemented tests.

A high-level overview of the platform is depicted in Fig. 2. Normalized alerts (in IDMEF-format) are accepted by the *verification manager*. The alerts are passed to various *test components*. Each test component attempts to verify a single aspect of the alert to measure the likelihood that it corresponds to a true positive and assigns a partial plausibility to the alert. This partial plausibility is also a floating point value in the interval $[0, 1]$, but can be simply 0 or 1 in the case of a boolean test. A test is not required to produce relevant results in every situation. For example, the existence of vulnerabilities can not be verified if the alert does not contain vulnerability references. In this case, a test component is able to explicitly state that it is unable to verify the alert. A description of the test components follows:

Sensor verification: the sensor verification test tries to verify the reliability of the sensor that generated the alert. A sensor may be badly configured or compromised. By applying anomaly detection methodologies on the behavior of the sensor and optionally correlating these results with the behavior of other sensors, configuration problems or misbehaving sensors can be detected. The test returns a partial plausibility approaching 0 for an unreliable sensor, or 1 for a reliable one.

It is also possible to take into account the confidence an administrator has in a sensor. For example, an administrator is able to express more trust in a hardened NIDS

that has been carefully tuned. On the other hand, an experimental HIDS on a non-hardened host will be less trusted. By combining a static measure of trust in each sensor, and observing alerts of attacks that target the originating sensor (taking into account their plausibility), we derive a measure of confidence in the integrity of the host of the sensor and whether the sensor should be trusted.

Reachability check: this test attempts to check if the detected attack can reach the target. If the asset database contains a model of the network, a passive check can be made to ensure that the detected attack should be able to reach the intended target from the point of detection, according to the model. This approach can filter out alerts of attacks targeted at non-existing hosts.

The reachability check can also verify configurations of firewalls, proxies etc. to ensure that the attack is able to reach the intended target. This info can be passively gathered from the configuration of these devices, or actively obtained using tools such as *firewalk* (http://www.packetfactory.net/projects/firewalk/) among others, from the location of the detecting sensor. In this test, the plausibility is discrete: $p = 1$ if the target is reachable for the specified protocol and port, or $p = 0$ otherwise.

Vulnerability verification: an important facet to distinguish true positives from non-relevant ones, is the vulnerability of the target to the detected attack. This test attempts to verify just that. Vulnerability data can be obtained using vulnerability scanners such as *Nessus* (http://www.nessus.org/). This test also produces a discrete result: $p = 1$ if the target is vulnerable, $p = 0$ otherwise.

Target check: the last test component attempts to verify the target of the detected attack. Three aspects of the alert are verified: is the target host responding normally, are the services offered by the target host responding normally and have any related alerts been detected?

The first two parts of the test check if the host is still responding, that all services that should normally be provided are still responding as expected and that no new services are found. The rationale behind this is that failed exploits sometimes crash a service or even an entire host. When a host is compromised, however, usually back doors or illegal services are installed. Information from the asset database is compared with results from an nmap scan of the target: the services that the target should offer are compared with those found. Anomalies result in $p = 1$, otherwise $p = 0$.

The third part of the test is able to verify if other, related, alerts have been detected. In the ideal case, the verification platform has a database with preconditions for each attack (similar to the pre- and post-condition based correlation approaches), so that possible preludes to an attack can be taken into account when verifying an alert corresponding to this attack. In our simplified approach, we only take the plausibility of previous attacks targeted to the same host into consideration. If prior alerts with a high plausibility have been detected, the maximum of this plausibility and the results of part one and two of the test is returned.

Note that, although this test may benefit from the same information as certain correlation approaches (a list with pre- and postconditions for each attack), the purpose of this test is not to correlate alerts. The information can be used by the verification

component to verify if all preconditions necessary for a certain attack have been fulfilled, besides allowing the correlation of this attack with other alerts in the correlation component.

The test results are passed to the *results manager*. The results manager combines these intermediate results using a *results algorithm* into a single floating point value. Two categories of tests can be considered: tests that validate necessary conditions (e.g. reachability and vulnerability of the target) and tests that check other indications of possible falsehood or non-relevance of the alerts (e.g. compromise of the detecting sensor or anomalous results from the target check). While the latter provide extra contextual information, the former should be satisfied. The asset database is updated with newly gained asset information (from active scans by the verification tests, for example), in the *context update*.

The framework is extensible in the following ways: first of all, new test components can be added easily at runtime. An example of an extra test is mentioned in related work (see Section 4). Another adjustable factor is the algorithm used to compute the final plausibility of alerts. While some algorithms are already provided (return the minimum or maximum of the results of the test components, compute a weighted average), new algorithms can be easily implemented and added. A third extensible factor is the asset database: new contextual information on the monitored infrastructure is added easily, enabling a more contextually aware alert verification.

3.2 Proof-of-concept Implementation

We have implemented the platform in Java, using a MySQL asset database, the Connector/J JDBC implementation and Apache Xerces to parse IDMEF alerts. The correct operation of the platform is validated in a test setup with some sample attacks. Multiple approaches are possible to evaluate the correctness and efficiency of the verification process, e.g. sets of off-line test data, like the DARPA sets[3], deploying the verification platform in a simulated environment or use it in a live test setup. The active nature of certain tests excludes the use of off-line test data, however. Therefore, preliminary tests of the framework have been performed in a test setup.

To illustrate our verification approach, test results of a sample alert are presented. In our setup, a web server is protected by a firewall. Two network-based sensors (one in front of the firewall and one behind it) and one host-based sensor on the web server monitor the infrastructure and relay their alerts, in IDMEF format, to the verification platform. No further correlation is done. An alert is generated for a trinoo attack[4]. More specifically: an attempt to contact a trinoo daemon is detected.

In the setup, the trinoo connection was blocked by a firewall protecting the web server. Consequently, the target is marked as being unreachable for the attack and the reachability test returns 0. The confidence in the detecting sensor is 0.7. No irregularities are detected by a port scan of the target and no other alerts have been detected,

[3] http://www.ll.mit.edu/IST/ideval/data/data_index.html

[4] Trinoo is a distributed denial-of-service attack. Trinoo daemons are installed on compromised hosts. Afterward, a trinoo-master is able to remotely order these compromised hosts to execute a denial-of-service attack against a specified target.

Table 1. Verification results for the trinoo alert.

	sensor	target	reachable	vulnerable	final result
p	0.7	0.0	0.0	NO_RESULT	0.107

therefore the target check returns 0. The vulnerability check does not produce results, as the trinoo alert does not contain references to a vulnerability. A weighted combination of these results produces a final plausibility of 0.107, marking this alert as being likely false or non-relevant.

The quality of the results is highly dependent on the available asset information. If no (or insufficient) asset information is available, alerts are automatically given a high plausibility to prevent the system from marking true positives as false. Because of this, it is still possible that false or non-relevant positives receive a plausibility that is indistinguishable from true positives, limiting the usefulness of verifying these alerts.

4 Discussion and Related Work

One of the major problems of intrusion detection is, as mentioned in Section 1, the inability of (simple) IDS to distinguish relevant alerts from non-relevant ones. In essence, the IDS is insufficiently context-aware to make this distinction. Verification raises the context-awareness of the intrusion detection process, by eliminating these non-relevant alerts.

This context-awareness—extra information on the monitored infrastructure—is a necessity to solve the intrusion detection scalability problems. The success of intrusion detection is dependent on the quality of asset information, be it network topology information, vulnerabilities etc. As we have seen, this information is used by the verification and correlation components. Also, extra information on attacks, like their pre- and post-conditions, is required for some verification tests and correlation methodologies.

As such, alert verification and correlation do not completely solve the intrusion detection problems, but shift them to the gathering of accurate information on the protected infrastructure and possible attacks. The accuracy of intrusion detection is proportional to the quality of this information. The verification framework allows to make the trade-off between performance and quality of available contextual information for the verification process in a centralized and configurable fashion: the asset data could be updated reactively, in response to a request for verification of an alert. Conversely, proactive scans could keep the asset information up-to-date at all times. In other situations, active scanning could be disabled altogether. The asset data could still be updated by relying on passive techniques or information entered by the administrator.

Currently, there is no consensus on what the correlation process is and how it should be implemented [1]. Multiple correlation strategies exist, for example: probabilistic alert correlation [4], the STAT-framework [6], approaches developed by Ning et al. [7, 5, 8], the CRIM-module of the MIRADOR project of the French defense department [9] and statistical causality analysis [10]. A proposition for a general correlation model is made in [1].

There is also no definitive answer to how verification should be performed. Ideas mentioned in other works include vulnerability scanning [1], checking network topology and firewall configurations [11, 1], verifying the behavior of the target host and service [1], administrator preferences [4] and verification based on the absence of alerts from other sensors that should have noticed the attack [2].

This last approach, as described in [2], uses a model of the network to infer and compare two sets of IDS: the set of sensors that did detect the attack and those that should have detected the attack, but did not—the potentially reactive set. Based on this information, it is possible to distinguish false positives from true positives. This approach is not able to detect the non-relevance of alerts, however.

Our work complements most of the above mentioned work. We focus on verification and enable the integration of the plausibility that an alert is a true positive as a factor in the correlation process. Our future work on alert verification will include searching for a more optimal results algorithm to compute the actual plausibility of an alert, based on intermediate results from the test components, and compiling a more exhaustive list of verification tests.

5 Conclusion

Intrusion detection suffers from scalability problems: IDS generate a large number of alerts containing lots of false and non-relevant positives in the alert stream, the alerts are of an insufficient semantic level and the IDS still miss certain attacks. While these issues can not be solved by deploying more intrusion detection systems alone, they can be handled in combination with alert correlation. Verification, as an intelligent filter before the actual correlation algorithm, plays an important role by filtering out false and non-relevant alerts.

While general architectures for correlation and multiple correlation methodologies have been proposed, no generic framework for verification exists. We have developed an extensible generic framework for alert verification that allows for the integration of different verification tests, different results algorithms and contextual information on the protected infrastructure, in order to enable experimentation.

Our contribution is to verify certain aspects of alerts, both necessary for the success of the attack (i.e. is the target reachable and vulnerable) and aspects incorporating other contextual information (i.e. is the sensor reliable and the target behaving abnormal). Our first experiences show that our approach enables a more effective distinction between false and non-relevant positives on the one hand, and true positives on the other hand.

References

1. Valeur, F., Vigna, G., Kruegel, C., Kemmerer, R.A.: A comprehensive approach to intrusion detection alert correlation. IEEE Transactions on Dependable and Secure Computing (2004)
2. Morin, B., Mé, L., Debar, H., Ducassé, M.: M2d2 : a formal data model for ids alert correlation. In: Proceedings of the 5th symposium on Recent Advances in Intrusion Detection (RAID 2002). (2002)

3. Debar, H., Curry, D., Feinstein, B.: The intrusion detection message exchange format. Technical report, IEEE (2004)
4. Valdes, A., Skinner, K.: Probabilistic alert correlation. In: Recent Advances in Intrusion Detection (RAID 2001). Number 2212 in Lecture Notes in Computer Science, Springer-Verlag (2001)
5. Ning, P., Xu, D., Healey, C.G., Amant, R.S.: Building attack scenarios through integration of complementary alert correlation methods. In: Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04). (2004) 97–111
6. Vigna, G., Kemmerer, R.A., Blix, P.: Designing a web of highly-configurable intrusion detection sensors. Lecture Notes in Computer Science **2212** (2001) 69+
7. Ning, P., Xu, D.: Learning attack strategies from intrusion alerts. Technical report, NC State University (2003)
8. Ning, P., Xu, D.: Hypothesizing and reasoning about attacks missed by intrusion detection systems. ACM Transactions on Information and System Security (2004)
9. Cuppens, F., Miège, A.: Alert correlation in a cooperative intrusion detection framework. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy. (2002)
10. Qin, X., Lee, W.: Statistical causality analysis of infosec alert data. In: Proceedings of The 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003). (2003)
11. Chyssler, T., Burschka, S., Semling, M., Lingvall, T., Burbeck, K.: Alarm reduction and correlation in intrusion detection systems. (2004)