A test of intrusion alert filtering based on network information

Teodor Sommestad, Swedish Defence Research Agency (FOI). Linkoping, Sweden (corresponding author)

Ulrik Franke, Swedish Defence Research Agency (FOI), Stockholm, Sweden

Abstract: Intrusion detection systems continue to be a promising security technology. The arguably biggest problem with today's intrusion detection systems is the sheer number of alerts they produce for events that are regarded as benign or non-critical by system administrators. A plethora of more and less complex solutions has been proposed to filter the relevant (i.e., correct) alerts that signature based intrusion detection sensors produce. This paper reports on a test performed to test a number of filtering alternatives that take advantage of information about static properties of the monitored computer network, such as vulnerabilities and exposure of ports and hosts. The results show that none of the filters are able to maintain a high recall (portion of detected attacks) while increasing the precision (portion of relevant alerts). At most, precision increased from 1.4 percent to 2.9 percent, and this also resulted in a decrease in recall from 44 percent to 26 percent. Even when combined in an exploratory fashion the filters fail to provide improved precision. It is concluded that filters based on static properties of the computer network do not result in clear improvements to alert-lists produced by signature based intrusion detection systems.

Keywords: intrusion detection, false positives, alert filters

1 Introduction

Intrusion detection systems (IDSs) have been, and continue to be, a promising security technology that interests both practitioners and scientists. An IDS can serve both as a real-time tool to prevent ongoing attacks and as a support to incident analysis performed after-the-fact.

Overall, the research community has been focused on what Axelsson [1] refers to as anomaly based IDSs and there is no question that many such solutions have been proposed in the literature. The main alternative to anomaly based IDSs is the type of solution that Axelsson [1] calls signature based. As opposed to the research literature, the market is dominated by signature based network IDSs. A signature based network IDS has a straightforward design. It compares the network traffic to a set of signatures associated with attacks and raises an alert if the network traffic looks like

one of the attacks. The dominance of signature based solutions is indicated in an analysis by the technology analysis firm Gartner, noting that signature quality remains the primary selection factor on the market for IDSs with preventive capabilities (i.e., intrusion prevention systems) [2].

Given the extensive use of signature based intrusion detection systems, it makes sense to ask how the signature based solution should be employed to be as effective as possible. Because, in spite of being straightforward, signature based solutions are not unproblematic to employ in the real world. Werlinger et al. [3] and Goodall et al. [4] found that, in practical applications, considerable system administrator expertise is required to configure filters in order to make the rules in the intrusion detection system effective. However, few reliable empirical tests can be found on how effective signature based IDSs are at detecting attacks, or how their effectiveness can be improved.

This paper provides some answers related to the question of how a signature based solution can be improved using static information about the monitored system. More precisely, it is tested how the following information can help to filter the alerts produced by an IDS:

- internal IP-addresses
- open ports
- presence of known vulnerabilities
- vulnerabilities' base score (severity)
- vulnerabilities' access vector
- vulnerabilities' access complexity
- the level of authentication needed to exploit the vulnerabilities.

The underlying idea is to filter out those alerts that match actions of a presumed and rational attacker, e.g., to remove alerts that cannot be associated to an exploitable vulnerability. The filters are meant to be intuitive and should be straightforward for a practitioner to implement in their environment. As will be described in section 2, some of them have been previously proposed in the literature.

The filters, and combinations of them, are evaluated by applying them to the output produced by the commonly used intrusion detection system Snort on data produced during a cyber-security exercise held in 2012. The filters' impact are measured by comparing how they influence the number of alerts produced, the number of alerts raised because of an attack, (true positives), the number of alerts raised without an attack (false positives) as well as the information theoretic metrics called precision, recall and F1-measure.

The outline of this paper is as follows. Section 2 describes related work and previous tests. Section 3 describes the test filters. Section 4 describes the study design. Section 5 presents the results. Finally, in section 6, the result is discussed and conclusions are drawn.

2 Related work

A considerable number of research projects have been directed towards IDSs. It is safe to say that the majority of these projects have resulted in new or improved ways of determining whether attacks occur, typically by introducing new ways of classifying network traffic as benign or malicious. This section will focus on such previous work that (i) addresses programmable signature based intrusion detection systems and (ii) aims at filtering the alerts they produce using *static* information about the monitored system, i.e., information that does not change very much over time. Section 2.1 describes proposed information for filtering and section 2.2 describes quantitative evaluations reported in the literature.

2.1 Information used by filters

One suggested way to filter or prioritize alerts is to compare *installed software* with the software products mentioned in the alert [5][6] [7] [8] [9] [10]. For instance, if an alert concerns Linux-exploits but is raised for a Windows-machine it could be discarded.

A requirement for a successful attack is that the targeted system has a *vulnerability* that can be exploited. In the basic case only alerts that match a vulnerability known to be present in the targeted system is allowed through the filter ([11] [12] [13] [14] [7] [6] [15] [9]). More elaborate variants include prioritizing alerts based on the potential impact of the vulnerability [14] or the priority match between vulnerability and alert [6].

For a network attack to be performed it is necessary that the attacked software is exposed to the network. Consequently, the *network exposure* of the system or its vulnerabilities to the outside is sometimes included in filters. Open ports as well as IP addresses of machines and alerts are often used [8] [8][15] [9]. Filtering could also be done probabilistically based on network topology, where some topologies give some alerts a higher probability of being true positives [16].

The requirements that alerts should refer to refer to *installed software* with a network-*exposure* and that it should have a *vulnerability* matching the alert have also been combined in various ways (see for example [9] [17]).

It should be noted that some of the references above do not propose filters, but rather that information should be combined into a prioritization or relevance-rating. Furthermore, several of the proposals are meant to be used

for alert verification. Alert verification aims to remove all alerts that cannot lead to a system compromise, i.e., it also aims to discard real attack attempts that will not succeed. While verified alerts certainly are critical to know of, most system administrators would also want to know if someone has attempted to attack their system but failed. Thus, the filters investigated in this article aim to filter out all alerts based on real attacks, but the purpose of them is to increase precision of the system, i.e., to reduce the number of false positives that the system administrator needs to manage.

2.2 Quantitative evaluations

Seven quantitative evaluations of filters are summarized in Table 1. Two used variants of the Darpa dataset, one used the dataset described by Massicotte et al. [18] and the other four have used custom built their own test cases. Different criteria are used, e.g., precision (portion of relevant alerts), portion of correctly discarded/kept alerts, the portion of alerts reduced, difference in rate of true positives and difference in rate of false positives.

Performance improvements are in some regards impressive. However, it is questionable whether these results can be generalized to real-world scenarios. For instance, the improvements in precision observed by Bolzoni et al. [19] are for a system without users producing events, and the representativeness of the results reported by Waita et al. [12] from tests on the Darpa dataset (anno 1999) may not be generalizable to todays' computer systems.

The use of different criteria also makes it difficult to combine the results to produce generalizations about the effectiveness of these filters or compare them. For instance, Neelakantan and Rao [13] assess the vaguely defined "useful alerts"; the number of false positives accidently filtered in the test made by Waita et al. [12] is not reported; only Gagnon et al. [7] report false alerts from tests with background data involving simulated computer usage. Nevertheless, the results of these tests are promising and they suggest that there is a real potential in rather simple filters taking advantage of information about the target system. The test described in this paper aims to clarify the potential of different alert filters.

3 Tested filters

A total of 18 filters were tested. The underlying idea and the information used in these filters are similar to the tests described in section 2. They used the following information to filter out relevant alerts: owner of IP-addresses, open ports on machines, existence of vulnerabilities and the following attributes from the Common Vulnerability Scoring System (CVSS) [20]: base score, access complexity, access vector and authentication. Table 2

gives a reference to the information used and a textual description of each filter.

The idea behind filter 1 is that attacks typically are performed from external, untrusted, machines. Thus, if an alert does not mention an external machine, the probability that it is false ought to be higher than usual.

Several of the filters rest on the assumption that attackers are attracted to machines that are vulnerable, and the more vulnerable the machine is, the more attracted they ought to be. Filter 2 removes all alerts where the targeted machine lacks a vulnerability and filters 3-6 removes alerts where the targeted machine lacks a vulnerability with properties that ought to be desirable for attackers. The CVSS base score reflects the overall severity by aggregating characteristics that are constant over time, independent of the particular environment. For example, a vulnerability is classified as high severity if it can be exploited over a network without authentication credentials or if it can be exploited over a network and has low access complexity. Filters 4-6 are more specific:

- Filter 4 requires that the machine has a vulnerability that can be exploited over a computer network.
- Filter 5 requires that the machine has a vulnerability with access complexity low. Low access complexity means, for example, "that affected configuration is default or ubiquitous" and "the attack can be performed manually and requires little skill or additional information gathering".
- Filter 6 requires that the vulnerability can be exploited without authentication. In other words, the attacker does not have to be logged in to the system to be able to exploit the vulnerability.

The remaining filters address the ports mentioned in the alert. Filter 7a requires that the ports that are mentioned are open while filter 7b requires that the alert mentions an open port and removes alerts not mentioning ports. Filters 8a-12b have the same type of vulnerability requirements as described above with the additional requirement of an open port. In other words, the software listening to the port must also have a vulnerability (8a-8b), or a vulnerability with certain attractive characteristics (9a-12b).

4 Study design

The 18 filters presented in Table 2 were tested based on a November 2012 exercise. The test involved the following steps:

1. Preparation of the monitored system environment.
2. Generation of background data and events.
3. Selection an attack scenario and injection of attacks.

4. Configuration of detection sensors
5. Encoding events and alerts
6. Analysis of effectiveness

In the sections below each of these are described in further. The files related to this test have been made publicly available (see [21]) and the authors of this paper can provide additional details concerning parameter setting and the study design.

4.1 Preparation of the monitored system environment

In this test, computer networks were instantiated within CRATE, the cyber range of the Swedish Defence Research Agency (FOI). Over a thousand virtual machines were deployed, together forming computer networks of various size and complexity. Of these, 153 machines in nine fictitious organizations of different type were monitored and considered targets (cf. section 4.3). Some organizations' networks consisted of a few computers to represent a small organization or personal network; other organizations' networks consisted of several VLANs with firewalls limiting access possibilities between them. Figure 1 illustrates the routing infrastructure in the synthetic environment and Figure 2 illustrates one of the monitored computer networks.

A number of different operating systems and applications were instantiated in these networks. Different patch levels and versions of Windows (2000, XP, 2003, 7, 2008) and a number versions of Linux-based distributions (Gentoo, Debian, Ubuntu) were used. These ran a number of desktop applications and server applications. Among other, the client-side applications included different versions of Adobe Reader, software development tools like Visual Studio, web browsers like Internet Explorer or Firefox, and applications of the Microsoft Office suite or Open Office. Server-side applications included different versions of Wordpress, phpMyAdmin, IIS, Domain Controllers, network infrastructure services (e.g., DNS and DHCP), and FTP servers. The aim was that the deployed applications should be representative of the standard software found in most enterprise computer networks. However, to enable a meaningful exercise for the attacking teams and to produce enough data for the test, these applications were more vulnerable than the ones found in the typical enterprise (i.e., they had not been updated and patched recently). Also, custom built applications (e.g., interconnected spreadsheet applications) and larger enterprise systems (e.g., ERP systems) were not present.

4.2 Generation of background data and events

An essential component in a test addressing precision and false positives of an IDS is the benign background traffic. Without background traffic, detection of attacks is trivial (every event is an attack). Background traffic is

thus essential to the problem: since the vast majority of the traffic in an organization is benign, even a small share of false alerts will be significant in absolute numbers and cause a problem for the system administrator [22]. Two main alternatives are available to produce background traffic: recording from real networks or simulating synthetic events [23].

Recordings from real networks have the advantage of providing realism. However, there are a number of problems associated with using real data for research [23]. First, it is non-trivial to obtain permission to inject attacks in real operational systems since they can cause disturbances. Second, confidentiality constraints make it difficult to share the data within the research community, and thus difficult to the requirement of repeatability associated with the scientific process. Third, there is no guarantee that the recorded background traffic is free from malicious acts that an IDS should raise an alert for. In addition, for the type of test performed in this paper a fourth problem exists. Since the test aims to investigate how vulnerability scanners can improve the precision and recall of an IDS it is of essence that the monitored systems actually contain a considerable number of exploitable vulnerabilities. Thus, a real environment with questionable security management is needed – but as per the third requirement, it must nevertheless be free from attacks not inserted by the researcher.

Simulated synthetic events and simulated traffic is free from these problems. However, it offers no guarantee of being realistically close to any particular environment and is therefore problematic to make generalizations from. In this test, background data was produced synthetically by producing events through the software applications installed on the client machines. To provide more realism, the events were produced based on recordings made in real systems.

The events were produced with scripts implemented in Auto IT [24] to emulate user actions by using installed applications to send emails to each other, surf websites in the environment, open emails/attachments and access files on local machines.

In order to produce a realistic behavioral pattern, the emulated users performed these behaviors according to a predefined instruction list created based on the actions of real users in an office environment. The instruction list was produced by collecting the historic usage of web browsers, desktop applications and the outgoing emails of 17 individuals in three organizations (two research organizations and one game developer). To enable a variety of user behavior the historic records (covering months to years of computer usage) were split into one-week instruction sets, which allowed thousands of instruction lists (i.e. “users”) to be created. Since the real users’ usage of desktop applications (e.g., the websites they visited) lacked meaning in the

isolated environment of this test, the user agents employed in this test translated each website, user and file into something meaningful in the test environment. For instance, a query made on *www.google.com* was interpreted as query made on *www.boogle.ex*, the search engine of the fictive test environment that returns a result meaningful in this environment. Thus, the activities performed by the scripted user agents followed the same sequence and had the same intensity as real users do during a work week. However, the content of mail, files and websites they accessed are unlikely to be representative of the content the real users accessed. Also, the scripted users did only perform standard behavior of users – more sporadic tasks like installation of custom software applications or administrative tasks were not performed in this test.

4.3 Selection of an attack scenario and injection of attacks

In this test two independent teams attacked the computer networks in order to find secret keys hidden in them. One team consisted of security researchers from the Swedish Defence Research Agency, the other team consisted of security specialists from the Swedish Armed Forces Network and Telecommunications Unit. Both teams restricted themselves to using only publicly available tools and publicly known exploits, e.g., the tools and exploits packed with the operating system Backtrack 5.

The attacks started at noon a Tuesday and continued until noon a Thursday. The attackers had no prior knowledge about which machines the secret keys were hidden in, but were told that they were in some of the nine monitored networks. As a result, a mix of reconnaissance and penetration activities was conducted. According to their own activity logs the two teams conducted 278 penetration attempts and network scans (or other reconnaissance related activates) aimed at the network monitored in this test. These penetration attempts led to the compromise of 98 machines.

During the test, a system administrator monitored the status of the systems and the intrusion detection alerts in real time. When this administrator saw obvious and loud intrusions he responded and tried to revoke the access credentials obtained. As a result, some attacks are similar to each other as the attackers then tried to regain the revoked credentials.

4.4 Configuration of detection sensors

Nine networks were monitored in this test. In these networks, all traffic was monitored using Snort 2.9.0.5 running a snapshot of the full rule set dated March 8, 2011.

Snort was configured with the default rule set to ignore alerts of priority 0 (“None”) and priority 4 (“No priority”). In addition, the following rules were ignored because of the numerous false alerts they produced in the test

environment: SID 129-4 on incorrect timestamps in packets, SID 129-12 triggered by small packets, SID 399 on unreachable hosts and SID 3218 on machines trying to connect to a server using Microsoft RPC DCOM.

During the time period of this test, a total of 624,218 alerts were generated by Snort. The analysis and comparison to attack logs show a somewhat stochastic behavior where the same type of action by the attackers generated different responses from Snort (occasionally no response at all). One possible cause of this stochastic behavior is dropped packets due to performance problems. While this is possible, the stochastic behavior seems uncorrelated to periods of high network load. Also, the hardware used in this test is well above the recommended minimum. At most the sensors received 3.5 Gigabit per hour (on average 1 Mbit/s), while [25] states *"A very rough and conservative rule of thumb is that Snort running on a single CPU can examine 200Mbits/sec of traffic without dropping an appreciable number of packets"*. Thus, it appears unlikely that capacity problems caused the stochastic behavior and more likely that other factors (e.g., features of the attackers' actions) did.

4.5 Encoding events and alerts

To assess the impact of the filters, the number of detected attacks and the number of false alerts were assessed. From these, the precision (fraction of alerts that are due to an attack) and the recall (fraction of attacks that are detected) were derived.

Logs manually created by the attacking teams were used to identify attacks and scans produced. These logs were used to mark each recorded alert as either correct (i.e., correctly indicating an ongoing attacker's activity) or incorrect (i.e., unrelated to attackers' activities) post hoc. All other alerts were treated as incorrect, i.e., false positives. Thus, only those actions that the attackers regarded as attacks were seen as real attacks. This is obviously a crude interpretation, although it is difficult to identify a better alternative. Comparisons between screen recordings from the attackers computers does for instance show that they did not regard use of compromised machines as an attack in itself and they seldom regarded browsing public services or sending PING request to public services as attacks.

The recorded alert was coded as correct true positives if it was seen as plausible that an operator receiving the alert would understand that the corresponding attack was ongoing. This means, for instance, that alerts related to PING-requests and overlapping TCP packets that followed the normal pattern and intensity were considered false positives even though they may stem from the attackers actions.

4.6 Analysis of effectiveness

Three of the filters are based on detailed information concerning the monitored computer network. More precisely, they require: network address, open ports/services, installed software products and security vulnerabilities. Internal and external network addresses was retrieved from configuration files used to deploy the machines. The other information was gathered using authenticated network scans with Nexpose. To our knowledge, the performance of Nexpose when it comes to true positives and false negatives is comparably good [26].

Intrusion alerts, attackers' logs and vulnerability information was structured in a relational database. The filters were tested by querying the database post-hoc and counting: total number of alerts that remained, alerts marked as correct and the unique number of attacks detected (an attack can raise multiple alert). From these the information theoretic metrics precision and recall and the F1-measure were calculated. Precision represent the probability that an alert is raised because of an attack, recall represent the probability that an attack raises an alert at all, and the F1-measure aggregates these as the harmonic mean of precision and recall.

5 Results

As shown in Table 3, all filters result in a decreased recall (portion of detected attacker actions) in comparison to the baseline with no filter.

As one would expect, less restrictive filters, like filter 2 which only require the existence of a vulnerability in the targeted host or 7a which requires that ports mentioned in the alert are open, reduce the recall the least. However, the ones that maintain highest recall (i.e. filters 2, 6, 7a and 8a) still produce a large number of alerts and a large portion of false positives. The large portion of false positives can be seen by the low precision they result in. They all produce lower a lower precision than the baseline. Thus, they tend to filter out a larger portion of true positives than false positives.

As Figure 3 illustrates, no filter performs strictly better that the baseline in terms of precision and recall, but several filters perform strictly worse (i.e., with both precision and recall that is lower than the baseline). The only filter leading to a considerable improvement in precision (to 2.9% from 1.4%) is filter 1, based on the owner of IP-addresses in the alerts. However, while this filter doubles the precision it reduces the recall by almost half.

Several of these filters are combinations of other filters, as Table 2 shows. In addition, the CVSS-severity score is an aggregate of filters 4, 5 and 6. These relationships explain why several filters produce identical results. For instance, filter number 4 and 5 produce the same results because all

machines with vulnerabilities of CVSS-severity seven and above also have vulnerabilities that can be exploited over a network.

These poor results lead to the question if other combinations of filters could possibly yield better precision and recall. Thus, after the test other meaningful combinations of filters were sought. Two types of combinations were seen as theoretically sound, i.e., they were believed to reflect the decision model and actions of attackers. First, it was considered possible that the aggregation of *Authentication*, *Access complexity* and *Access vector* made by the CVSS did not reflect the attractiveness of the vulnerability to attackers. Filters with an AND-combination of these three and all pairwise AND-combinations of them were tested. All these AND-combinations perform as filters 4 and 5. The OR-combination of *Authentication*, *Access complexity* and *Access vector* was also tested¸ based on the idea that a permissive state in any of these would attract an attacker. This OR-combination had a recall as almost high as when no filter is used, but a poor precision (0.7%). Another combination that seemed promising was to require filter 1 in combination with the other filters, i.e., to always require that and external host is involved in the alarm. However, none of these seventeen combinations produced a higher precision than filter 1 alone, but all produced a lower recall. Thus, none of these combined filters improves the filtering.

6 Discussion and conclusions

To offer control and make sure that operational systems were not impacted this test was performed in a synthetic environment using a fictitious scenario. As a result, there are a number of factors in this test with questionable ecological validity (i.e., realistic conditions). Among other things, it could be argued that the background traffic was not diverse enough, that the targeted networks were unusually small or that the attackers and their attacks were overly intense. These factors most probably influence the performance of the IDS as measured in this test and it can be argued that the performance of the IDS is exaggerated because of unusually beneficial conditions. For instance, if attacks would be stretched out over a longer period of time, with background traffic intensity remaining the same, it would certainly reduce the precision of the IDS. Thus, it is easy to identify factors that may have an influence on the efficacy of an IDS and question how they were represented in this test. However, for most of these factors it appears unlikely that they influenced the *relative* performance of the filters compared to the baseline with no filter.

In other words, most of the factors that influence the efficacy of an IDS ought to have the same (relative) impact on all tested filters. In fact, the only factors we can identify that are likely to have an impact on the relative

performance of the filters are the vulnerabilities present in the network and the attacks performed against the networks. Clearly, the vulnerability based filters would perform better if the information on relevant vulnerabilities would be more accurate. Thus, if a more competent vulnerability scanner was used or intelligence was available on vulnerabilities that attackers tend to exploit these filters ought to perform better. Conversely, if attackers possess exploits of zero-day vulnerabilities the performance of these filters would decrease. While vulnerability information and the attacks used ought to influence the results of this test we believe that this scenario, with a commonly used vulnerability scanner and publicly known attacks, represents a relevant and realistic case.

When it comes to the practical relevance of this study there are also potential issues associated with the encoding and analysis method used. As mentioned in section 4.5 the definition of an attack, and consequently a true positive, is crude but straightforward. Although most real IDS users probably consider some types of attack steps as more important to detect than other, no structured attempt was made to rate the importance of different actions in the attacker's log book and take this into consideration in the evaluation. For instance, if alerts on penetration attempts are valued differently than alerts on reconnaissance (e.g., network scans) the following indicates how the filters influence alert quality. Without filter, 42 percent of the penetration attempts and 43 percent of the reconnaissance activities raised alerts; with filter number 7b or 8b, 24 percent of the penetration attempts and 10 percent of the reconnaissance activities raised alerts. Thus, the filters are more sensitive to penetration attempts, but the difference is not dramatic.

While the ecological validity of this test can be questioned and a crude evaluation method was used, it can be concluded that filters based solely on static information about the monitored computer network do not solve the problem of false alerts associated with IDSs. With Snort and Nexpose, the best filter in terms of precision increased the portion of relevant alerts from 1.4 percent to 2.9 percent and this came at the cost of reducing the portion of detected attacks from above 44 percent to 27 percent. In absolute terms it is likely that the results overstate the efficacy of IDS-alternatives because of the high attack intensity and standardized background traffic. Still, the best filter raised 34 false alerts for every correct alert (2.9% precision) and only raised alerts for one attack out of four. Based on results in other fields than network security, Axelsson states that a very conservative estimate is that more correct than incorrect alerts (i.e., over 50% precision) is required to maintain system operator faith in a detection system [22]. The results obtained in this test makes it is hard to see how any filter based solely on

static information about the computer network can come close to this under somewhat realistic conditions.

7 Acknowledgments

This work has been supported by Security Link, in the Strategic Area for security and crisis management research, funded by the Swedish Government. Hannes Holm gave valuable remarks on the manuscript.

8 References

1. Axelsson S. *Intrusion detection systems: A survey and taxonomy*. *Technical Report* 99 -15. Göteborg, Sweden, 2000.

2. Young G, Pescatore J. *Magic quadrant for network intrusion prevention system appliances, Gartner RAS Core Research Note G00167309 2009, Gartner,* 2009.

3. Werlinger R, Hawkey K, Muldner K, Jaferian P, Beznosov K. The challenges of using an intrusion detection system: is it worth the effort? *SOUPS '08 Proceedings of the 4th symposium on Usable privacy and security,* 2008; 107–118.

4. Goodall JR, Lutters WG, Komlodi A. Developing expertise for network intrusion detection. *Information Technology & People,* 2009; 22; 92–108.

5. Bakar, N, Belaton B. Towards Implementing Intrusion Alert Quality Framework. In *First International Conference on Distributed Frameworks for Multimedia Applications*, 2005; 198–205. doi:10.1109/DFMA.2005.49.

6. Njogu HW, Jiawei L, Kiere JN, Hanyurwimfura D. A comprehensive vulnerability based alert management approach for large networks. *Future Generation Computer Systems*, 2013; 29; 27–45. doi:10.1016/j.future.2012.04.001.

7. Gagnon F, Massicotte F, Esfandiari B. Using Contextual Information for IDS Alarm Classifi□cation. In *Proceedings of International Conference, DIMVA*, Como, Italy, 2009; 147–156.. doi:10.1007/978-3-642-02918-9_9.

8. Meng Y, Li W. Constructing Context-based Non-Critical Alarm Filter in Intrusion Detection. In *The Seventh International Conference on Internet Monitoring and Protection*, 2012; 75–81.

9. Xiao M, Xiao D. Alert Verification Based on Attack Classification in Collaborative Intrusion Detection. *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, 2007; 739–744. doi:10.1109/SNPD.2007.216.

10. Hubballi N, Biswas S., Nandi S. Network specific false alarm reduction in intrusion detection system. *Security and Communication Networks,* 20122; 4: 1339–1349. doi:10.1002/sec.261.

11. Gula R. Correlating ids alerts with vulnerability information Technical report. Tenable Network Security. 2002.

12. Njogu, HW, Jiawei L. Using Alert Cluster to reduce IDS alerts. In *2010 3rd International Conference on Computer Science and Information Technology*, 2010; 467–471. doi:10.1109/ICCSIT.2010.5563925.

13. Neelakantan S, Rao S. A Threat-Aware Signature Based Intrusion-Detection Approach for Obtaining Network-Specific Useful Alarms. In *2008 The Third International Conference on Internet Monitoring and Protection*, 2008; 80–85.. doi:10.1109/ICIMP.2008.24.

14. Gupta D, Joshi PS, Bhattacharjee K, Mundada RS. IDS alerts classification using knowledge-based evaluation. *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*, 2002; 1–8. doi:10.1109/COMSNETS.2012.6151339.

15. Haukeli J. False positive reduction through IDS network awareness. Masters thesis, Oslo University College, Oslo, 2012.

16. Spathoulas GP, Sokratis KK. Reducing false positives in intrusion detection systems. *Computers & Security* 2010; 29; 35–44. doi:10.1016/j.cose.2009.07.008.

17. Chandrasekaran M, Baig M, Upadhyaya S. AVARE: Aggregated Vulnerability Assessment and Response against Zero-day Exploits. In *2006 IEEE International Performance Computing and Communications Conference*, 2006; 603–610. doi:10.1109/.2006.1629458.

18. Massicotte F, Gagnon F, Labiche Y, Briand L, Couture M. Automatic Evaluation of Intrusion Detection Systems. *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, 2006; 361–370. doi:10.1109/ACSAC.2006.15.

19. Bolzoni D, Crispo B, Etalle S. ATLANTIDES: An architecture for alert verification in network intrusion detection systems. In *Proceedings of the 21st Large Installation System Administration Conference (LISA '07)*, 2007; 141–152.

20. Mell P, Scarfone K, Romanosky, S. A complete guide to the common vulnerability scoring system version 2.0. *FIRST-Forum of Incident Response and Security Teams,* 2007, accessed 2014-06-13. http://www.first.org/cvss/cvss-guide

21. Sommestad T, Wedlin M. CRATE - Cyber Range And Training Environment. *Open datasets produced in CRATE*, Swedish Defence Research agency, 2012, accessed 2014-06-13. http://www.foi.se/en/Our-Knowledge/Information-Security-and-Communication/Information-Security/Lab-resources/CRATE/.

22. Axelsson S. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security, 2000;* 3; 186–205. doi:10.1145/357830.357849.

23. Athanasiades N, Abler R, Levine J, Owen H, Riley G.. Intrusion detection testing and benchmarking methodologies. *IWIAS 2003. Proceedings. First IEEE International Workshop on nformation Assurance. 2003;* 63–72. doi:10.1109/IWIAS.2003.1192459.

24. Bennett J. AutoIt Script Editor. AutoIt Consulting Ltd, 2013, accessed 2014-06-13. http://www.autoitscript.com/site/autoit-script-editor/.

25. Geekery. Capacity Planning for Snort IDS. accessed 2014--06-13, 2014. http://mikelococo.com/2011/08/snort-capacity-planning/.

26. Holm H, Sommestad T, Almroth J, Persson, M.. 2011. A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security* 19: 231–247. doi:10.1108/09685221111173058.

Table 1. Summary of quantitative evaluations of filters.

Study	Information used	Data used in evaluation	Result
[19]	• Installed software • Network exposure • Vulnerability	Snort alerts from a custom made testbed of six machines exposed to selected attacks launched with public tools.	Precision improved from 12-17% to 92-97%, depending on attack.
[7]	• Installed software • Vulnerability	Snort alerts from the dataset described in [18].	Almost all discarded alerts were non-critical (98-100%), depending on filter. The portion of non-critical alerts successfully classified as non-critical was 15% for the simplest filter and 73 % if software and vulnerability combination was considered,
[12]	• Network exposure • Installed software • Vulnerability	The Darpa datset and their own office network.	The number of alerts is reduced with 78% on the Darpa set and with 81% on their own network. The number of true and false positives lost is not reported.
[13]	• Vulnerability	A scenario with "various attacks" operating systems such as Windows 2000, Windows 2003 and Linux Red Hat.	The rate of "useful alerts" increased from 15-42% to 22-100%, depending on protocol.
[10]	• Vulnerability	A testbed with operating systems such as Windows 98/200/2003/XP, different Linux distributions services such as FTP and Telnet. No background traffic, but nearly 80% of the attacks were not matching vulnerability.	After training of the neural network the rate of alerts targeting an existing vulnerability increased from 20-30% to 90-100% with only some small drops in true positives (96-100% compared to 90-100%).

Table 2. Filter, information they use and a textual descriptions of them.

ID	IP-address owner	Open ports on machines	Existence of vulnerabilities	CVSS base score	CVSS access vector	CVSS access complexity	CVSS authentication	Alerts kept if
1	•							The alert involves an external host.
2			•					The targeted host in the alert has a vulnerability.
3			•	•				The targeted host in the alert has a vulnerability with CVSS base score of high (above 7).
4			•		•			The targeted host in the alert has a vulnerability with CVSS access vector network.
5			•			•		The targeted host in the alert has a vulnerability with CVSS access complexity low.
6			•				•	The targeted host in the alert has a vulnerability with where the level of authentication needed to exploit is none.
7a		•						No port is mentioned in the alert or the alert targets an open port.
7b		•						The alert targets an open port.
8a		•	•					No port is mentioned in the alert or the alert targets an open port where the associated software has a vulnerability.
8b		•	•					The alert targets an open port where the associated software has a vulnerability.
9a		•		•				No port is mentioned in the alert or the alert targets an open port where the associated software has a vulnerability with CVSS score of 7.0 or above.
9b		•		•				The alert targets an open port where the associated software has a vulnerability with CVSS score of high (above 7).
10a		•			•			No port is mentioned in the alert or the alert targets an open port where the associated software has a vulnerability with access vector of network.
10b		•			•			The alert targets an open port where the associated software has a vulnerability with access vector of network.
11a		•				•		No port is mentioned in the alert or the alert targets an open port where the associated software has a vulnerability with access complexity low.
11b		•				•		The alert targets an open port where the associated software has a vulnerability with access complexity low.
12a		•					•	No port is mentioned in the alert or the alert targets an open port where the associated software has a vulnerability where the level of authentication needed to exploit is none.
12b		•					•	The alert targets an open port where the associated software has a vulnerability where the level of authentication needed to exploit is none.

Table 3. Performance of the filters.

Filter	Number of alerts	True positives	Attacks detected	Precision	Recall	F1-measure
Baseline	624 218	8 819	123	1.4%	44.2%	2.7%
1	288 390	8 313	74	2.9%	26.6%	5.2%
2	502 162	4 137	116	0.8%	41.7%	1.6%
3	256 821	2 779	90	1.1%	32.4%	2.1%
4	17 678	158	8	0.9%	2.9%	1.4%
5	17 678	158	8	0.9%	2.9%	1.4%
6	479 636	3 475	114	0.7%	41.0%	1.4%
7a	569 548	6 807	108	1.2%	38.9%	2.3%
7b	40 407	777	48	1.9%	17.3%	3.5%
8a	569 548	6 807	108	1.2%	38.9%	2.3%
8b	40 407	777	48	1.9%	17.3%	3.5%
9a	227 718	1 550	77	0.7%	27.7%	1.3%
9b	22 849	398	29	1.7%	10.4%	3.0%
10a	228 040	764	45	0.3%	16.2%	0.7%
10b	17 678	158	8	0.9%	2.9%	1.4%
11a	228 040	764	45	0.3%	16.2%	0.7%
11b	17 678	158	8	0.9%	2.9%	1.4%
12a	425814	1 615	87	0.4%	31.3%	0.7%
12b	23209	399	29	1.7%	10.4%	3.0%