An Active Traffic Splitter Architecture for Intrusion Detection*

I. Charitakis*, K. Anagnostakis†, E. Markatos*

*Institute of Computer Science
Foundation for Research and Technology - Hellas
P.O.Box 1385 Heraklio, GR-711-10 GREECE
{haritak,markatos}@ics.forth.gr

†Distributed Systems Laboratory
CIS Department, Univ. of Pennsylvania
200 S. 33rd Street, Phila, PA 19104, USA
anagnost@dsl.cis.upenn.edu

Abstract

Scaling network intrusion detection to high network speeds can be achieved using multiple sensors operating in parallel coupled with a suitable load balancing traffic splitter. This paper examines a splitter architecture that incorporates two methods for improving system performance: the first is the use of early filtering where a portion of the packets is processed on the splitter instead of the sensors. The second is the use of locality buffering, where the splitter reorders packets in a way that improves memory access locality on the sensors. Our experiments suggest that early filtering reduces the number of packets to be processed by 32%, giving a 8% increase in sensor performance, while locality buffers improve sensor performance by about 10%. Combined together, the two methods result in an overall improvement of 20% while the performance of the slowest sensor is improved by 14%.

1 Introduction

Network Intrusion Detection is receiving considerable attention as a mechanism for shielding against "attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer network" [2]. The typical function of a Network Intrusion Detection System (nIDS) is based on a set of *signatures*, each describing one known intrusion threat. A nIDS examines network traffic and determines whether any signatures indicating intrusion attempts are matched.

Effective intrusion detection requires significant computational resources: widely deployed systems such as *snort* [5] need to match packet headers and payloads against tens of header rules and often many hundreds of strings defining attack signatures. This task is much more expensive than the typical header processing performed by packet forwarders and firewalls. Therefore, performing intrusion detection at high network speeds (e.g. 1 Gbit/s and beyond) requires the use of multiple sensors operating in parallel, fed by a suitable traffic splitter element.

Given the high resource demands of intrusion detection, we consider ways of boosting sensor performance by rethinking the design of nIDS traffic splitters. We argue that traffic splitters should implement more active operations on the traffic stream with the goal of reducing the load on the sensors, rather than just passively providing generic, flow-preserving load distribution.

This paper presents two such active mechanisms. The first is based on the observation that a significant fraction of packets only require header processing. Given that header processing is relatively cheap (and can be easily performed in hardware) we can implement this function as part of the splitter. The main benefit of this method of *early filtering* is that the amount of traffic that needs to be transmitted and processed by the sensors can be reduced significantly.

The second mechanism is based on the observation that different types of packets trigger different subsets of the nIDS ruleset, placing a significant burden on the sensor memory architecture (*i.e.* reducing memory access locality). We present an algorithm for *locality buffering*, so that packets of the same type are grouped together on the splitter before being forwarded to the sensors. The benefit of this method is that it increases performance without altering the semantics of the traffic stream and without requiring changes on the sensors.

2 Design

The system is composed of an early filtering element, a load distribution element and a set of locality buffering

*This work was supported in part by the IST project SCAMPI (IST-2001-32404) funded by the European Union. The second author is also supported by ONR under Grant N00014-01-1-0795.

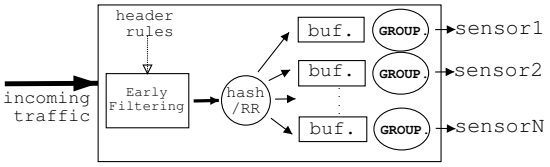

Figure 1. The active nIDS splitter architecture

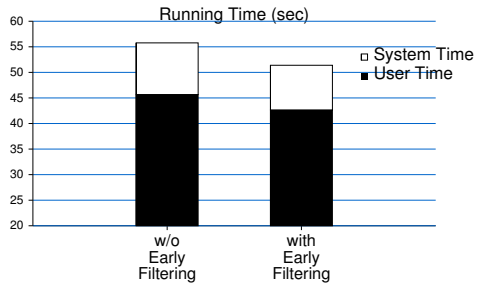

Figure 2. The effect of Early Filtering on sensor performance.

units, one for each sensor. This is depicted in Figure 1. In the remainder of this Section, we will present each of the elements. For further details please refer to [3].

2.1 Early filtering

The basic idea in early filtering is to implement part of the sensor functionality on the splitter. Since a fraction of packets is only subject to header analysis, which is significantly cheaper than content-matching, we can efficiently perform this function on the splitter. This is expected to reduce the load on the sensor but also on the overall system, as the process of sending packets from the splitter to the sensors can often be avoided.

To perform early filtering we analyze the nIDS ruleset and extract the rules that do not require content matching. We observe that this is a small portion of the default ruleset in *snort*: only 165 of 1700 rules. We refer to this set of rules as the *EF ruleset*. We expect that processing a small number of header rules in the EF ruleset on the splitter can be easily supported in hardware.

The splitter then operates as follows. When a packet is received it is first checked against the EF ruleset. If no rule is matched and the packet contains no payload then the packet is discarded. If no rule is matched but the packet does contain a payload that needs to be analyzed, it is forwarded to one of the sensors. If the packet matches a rule then again, it is forwarded to the sensors for generating an alert.

2.2 Locality buffering

Locality buffering is a method for adapting the packet stream in a way that improves performance of each nIDS sensor. The idea is based on the fact that there are specific rulesets for specific types of traffic. Consequently, when checking a packet, each sensor will have to bring the working set that corresponds to the rules of that packet. Alternating between rulesets results in cache misses due to conflicts and therefore performance degrades.

To increase memory locality, the splitter analyzes incoming packets, and then places them on separate buffers trying to keep in each buffer packets of the same type. When a buffer becomes full, all packets are transmitted to the target sensor in sequence. This increases the average number of same-type packets that are received by the sensor back-to-back. Therefore increases memory locality which in turn results in better performance.

3 Experiments

We present experiments examining the effect of early filtering and locality buffering on nIDS performance.

For most experiments we use a 1.13 GHz Pentium III processor PC with 8 KB L1 cache, 512 KB L2 cache and 512 MB of main memory. The host operating system is Linux (kernel version 2.4.17, RedHat 7.2). The nIDS software is *snort* version 2.0-beta20 compiled with `gcc` version 2.96 (optimization flags `O2`).

Most experiments are performed by reading a packet trace from disk, except for the early filtering experiments where traffic is received from the network (to capture the effect of early filtering on the network subsystem). In the later case we use a simple network with two hosts A and B and a monitoring host S. Host A reads the trace from file and sends traffic to host B (using `tcpreplay`) over a 100 Mbit/s Ethernet switch configured to mirror the traffic to host S. As the exact timing of trace packets has negligible effect on nIDS behavior, we simply replay the trace at maximum rate (link utilization was roughly 90%).

We use the `nlanr.MRA.1031627450` packet trace from the NLANR archive captured in September 2002 on the OC12c (622 Mbit/s) PoS link connecting the Merit premises in East Lansing to Internet2/Abilene [4]. As the trace only contains the header portion of each packet we had to add uniformly random payload data to create realistic traffic. (The use of random payloads for nIDS evaluation is shown in [1] to offer reasonably accurate performance estimates.)

3.1 Evaluation of Early Filtering

Analyzing the trace reveals that more than 40% of the packets do not contain any payload. Most of these packets are TCP acknowledgments and none of them is matched by the EF ruleset. These packets can therefore be dropped by the splitter during early filtering instead of forwarding them to a sensor for analysis. Only a small fraction of packets are actually matched by the EF ruleset (less than 1% of total packets) and are therefore forwarded to the sensor.

On each sensor, early filtering is expected to reduce the actual detection workload as well as the burden on the network subsystem for processing interrupts and bringing packets from the network interface to user space for processing.

To measure the effect of early filtering on sensor performance we measure the user and system time of running *snort* with the original trace as well as a trace that does not contain the packets that are dropped by early filtering. The results are depicted in Figure 2. We observe that user time is reduced by 6.6% (45.67 sec vs. 42.66 sec) while system time is decreased by 16.8% (10.1 sec vs. 8.7 sec). Considering both user and system time the results suggest an overall improvement of 8%.

3.2 Effect of LBs on nIDS performance

To investigate the benefit of using locality buffers we measure the total nIDS workload in terms of measured user time on each sensor as well as the workload of the most loaded sensor, given that traffic is not perfectly distributed.

We determine how performance is affected when using different numbers of participating sensors, as well as varying the number and size of locality buffers.

3.2.1 Effect of LB vs. number of sensors

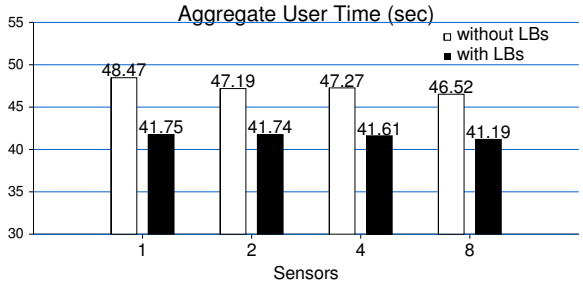

Figure 3. Aggregate user time over all sensors vs. number of sensors.

Figure 3 shows the aggregate user time for different numbers of sensors, and Figure 4 shows the the measured user time of the slowest (most loaded) sensor. For this set of experiments we use 16 LBs of 256 KB each. In all cases, using locality buffers improves the aggregate user time by at least 11.4% (8 sensors) and up to 13.8% (one sensor).

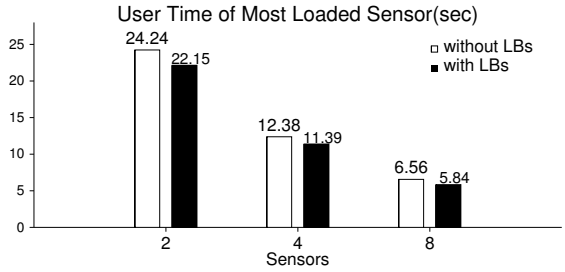

Figure 4. User time of slowest sensor vs. number of sensors for the experiments of Figure 3

One interesting observation from Figure 3 is that as the number of sensors increases, the effect of locality buffering is slightly decreasing. In fact, there is a slight improvement in aggregate user time even if we don't use locality buffering. This happens because distributing packets to different sensors demultiplexes the incoming traffic and increases the probability of same-type back-to-back packets. The positive effect of locality buffering is nevertheless evident.

To verify this observation we measure the average burst size (*e.g.,* the number of consecutive packets that have the same protocol and the same destination port) in the experiments of Figures 3 and 4. The results are presented in Figure 5.

We see that the average burst size is almost doubled when using LBs. We also observe that splitting traffic to more sensors slightly improves the mean burst size when not using LBs.

3.2.2 LB dimensioning

We investigate how the size and the number of locality buffers affect performance. In this set of experiments we use four sensors. In each experiment we measure the difference in user time compared to a system without LBs.

Figure 6 shows the results of using different number of LBs per sensor when the the size of each LB is 256 KB. We observe that the improvement in aggregate user time varies between 6.8% (4 LBs) and 12.9% (32 and 64 LBs). Increasing the number of LBs beyond 32 does not appear to offer any benefit in terms of aggregate user time although it still improves the performance of the most loaded sensor. This suggests that using 32 or 64 LBs is a reasonable design choice.

To measure how the size of each LB affects performance we measure, again, the aggregate user time and the user

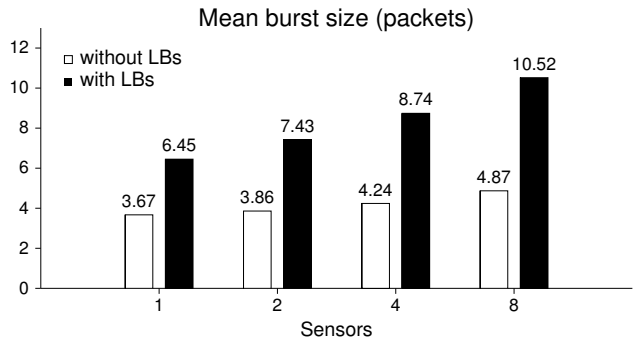

Figure 5. Mean burst size vs number of sensors for the experiment of Figures 3 and 4.

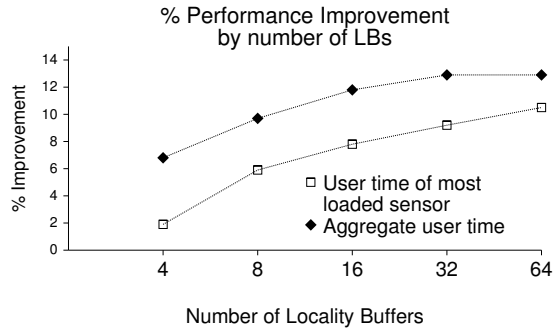

Figure 6. Performance improvement (reduction in user time) using different number of LBs.

time of the most loaded sensor for different LB sizes. The results are presented in Figure 7. The reduction in aggregate user time ranges from 9.3% to 13.31% for the cases of 64 KB and 512 KB respectively. Using 256 KB per LB seems like a reasonable choice, as the gain of increasing from 256 KB to 512 KB is marginal.

3.3 Evaluation of EF+LB

To estimate the benefits of using both early filtering and locality buffering together we apply the early filtering method on the packet trace and split the remaining packets to four sensors using 16 Locality Buffers of 256 KB per sensor. The measured aggregate user time is 37.88 sec compared to 41.61 sec when using LBs only, reflecting an improvement of 8.9%. Compared to 47.27 sec when not using LBs at all, the overall improvement of using both EF and LB is 19.8%. For the slowest sensor, performance is increased by 5% when compared to using only LBs (from 11.52 sec to 10.93 sec) and 14.4% when compared to not using EF or LB.

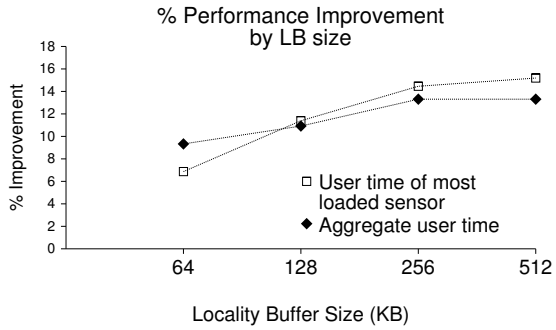

Figure 7. Performance improvement (reduction in user time) using different size for each LB.

4 Summary and future work

We have proposed an active traffic splitter architecture for intrusion detection. Rather than acting as a passive load balancing component, we argue that the traffic splitter should actively manipulate the traffic stream in a way that increases sensor performance.

We have presented and analyzed two specific examples of performance-enhancing mechanisms. The first is *early filtering*, where a subset of the traffic is processed on the traffic splitter and filtered out in order to reduce the load on the sensors. The second method is *locality buffering*, where packets classified to the same subset of nIDS rules are buffered together before being forwarded to the sensors. By grouping same-type packets and sending them to the sensor back-to-back, this method increases memory access locality on the nIDS sensors resulting in improved performance. When using both methods together, overall system performance is improved by 19.8%, while the running time of the most loaded sensor is improved by 14.4%.

References

[1] S. Antonatos, K. G. Anagnostakis, M. Polychronakis, and E. P. Markatos. Benchmarking and design of string matching intrusion detection systems. Technical Report 315, ICS-FORTH, December 2002.

[2] R. Bace and P. Mell. *Intrusion Detection Systems*. National Institute of Standards and Technology (NIST), Special Publication 800-31, 2001.

[3] I. Charitakis, K. G. Anagnostakis, and E. P. Markatos. An active traffic splitter architecture for intrusion detection. Technical Report 323, ICS-FORTH, July 2003.

[4] NLANR. MRA traffic archive, September 2002. *http://pma.nlanr.net/PMA/Sites/MRA.html*.

[5] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA Systems Administration Conference*, November 1999. (available from *http://www.snort.org/*).