This paper was presented as part of the 14th IEEE Global Internet Symposium (GI) 2011 at IEEE INFOCOM 2011

Improving the Performance of Intrusion Detection using Dialog-based Payload Aggregation

Tobias Limmer and Falko Dressler
Computer Networks and Communication Systems, University of Erlangen, Germany
{limmer, dressler}@cs.fau.de

***Abstract*—We propose Dialog-based Payload Aggregation (DPA) that extracts relevant payload data from TCP/IP packet streams based on sequence numbers in the TCP header for improved intrusion detection performance. Typical network-based Intrusion Detection Systems (IDSs) like Snort, which use rules for matching payload data, show severe performance problems in high-speed networks. Our detailed analysis based on live network traffic reveals that most of the signature matches either occur at the beginning of TCP connections or directly after direction changes in the data streams. Our DPA approach exploits protocol semantics intrinsic to bidirectional communication, i.e., most application layer protocols rely on requests and associated responses with a direction change in the data stream in between. DPA forwards the next N bytes of payload whenever a connection starts, or when the direction of the data transmission changes. All data transferred after this window is discarded. According to experimental results, our method reduces the amount of data to be analyzed at the IDS to around 3.7 % for typical network traffic. At the same time, more than 89 % of all potential events can be detected. Assuming a linear relationship between data rate and processing time of an IDS, this results in a speedup of more than one order of magnitude in the best case. Our performance analysis that combines DPA with Snort shows a 400 % increase in packet processing throughput on commodity hardware.**

I. Introduction

Attack detection on network traffic can be performed using methodologies from different classes. On the one hand, *anomaly-based* systems operate by constructing a traffic model with normal behavior using specific properties of the traffic. If a certain deviation from this model is observed, a so-called anomaly is detected. On the other hand, *rule-based*, or *signature-based*, systems have predefined rules that specifically look for certain properties in the observed traffic. If all properties of a rule are detected, it will be reported. High processing speeds can be achieved by using statistical properties that are retrieved from router statistics (e.g., the number of packets) or packet header data (e.g., packet sizes or traffic volume), as only a fraction of the packet data is processed. However, many common rule-based Intrusion Detection Systems (IDSs) use both packet header and payload data for event detection. This is also called Deep Packet Inspection (DPI). Thus, they need to process all data transferred over the network. Typical examples are network-based IDSs like Snort [1] or Bro [2].

The key concept is the following: First, if incoming packets represent a continuous data stream (e.g., a TCP connection), packets are reassembled to the original data stream using a module that checks the validity of each packet (i.e., the checksum and some connection-specific fields), verifies if the packet was transferred to the peer and the peer acknowledged its reception (to counter concealing attacks directed at the IDS) [3]. Afterwards, the IDS starts analyzing the incoming data: protocol-specific preprocessors analyze the data stream, extract information and save flags for later use. Then, a set of rules defines header and payload checks to determine whether some relevant event has been recorded. During this stage, patterns contained in the rules are matched to payload data. In some cases, these simple rules do not suffice and a complex state-based analysis needs to be performed. In these cases, the IDS either offers an adequate scripting language (e.g., Bro) or extension modules that process the traffic (e.g., Snort).

Payload-based pattern matching puts high resource requirements on the hardware. For current network speeds up to 10 Gbit/s and more, single IDS machines are not able keep up [4]. Besides parallelization, multiple approaches have been proposed used to speed up data analysis: *Filtering based on header data* is based on efficient flow aggregation [5]. Almost all header data (e.g., source and destination addresses, ports, packet size, protocol information and flags) may be checked by rules to filter data streams before payload is analyzed. The *use of efficient rules* is important, as, besides simple string matching, IDSs often support regular expressions for pattern matching. Finding a match for regular expressions may take up to one second in extreme situations [6]. To counter performance problems, regular expressions are often only evaluated if a simple content match was successful. *Optimization of pattern matching algorithms* is a common objective in computer science. Many new algorithms have been proposed in recent years [7]. The *use of specialized hardware* is another option. There have been several efforts that use specialized hardware for speeding up the payload matching process, like graphic cards or FPGAs [8], resulting in speed-ups of up to 60 %.

A completely different approach is *data sampling*, where the amount of input data is reduced by various lightweight selection mechanisms. As packet sampling would lead to incorrect per-flow information, *flow sampling* has been introduced to obtain this information [9].

In this paper, we present a technique for speeding up network-based IDS by performing intelligent and lightweight filtering on the input data, thus reducing the amount of data that needs to be processed by the IDS. Several methods for this type of payload aggregation have already been proposed in literature [10], [11], but to our knowledge, the *relevance* of the filtered data for security-related analysis has not yet been evaluated. Based on this evaluation, we propose a completely new method of data aggregation specialized on payload-based intrusion detection: Dialog-based Payload Aggregation (DPA). Filtering cannot be done in an IDS-independent manner, because we need to ensure that as many of the security-relevant events as possible are still detected. *Time machine* is one approach to the problem that was suggested in [10]. It stores the first *N* bytes of data per connection to enable security analysts to look at historical network data. We adapted this approach to live analysis that we named Front Payload Aggregation (FPA) [11] using extended IPFIX flow data [5]. For FPA, we coupled our monitoring-framework Vermont [12] with Snort and only forwarded the first *N* bytes of each connection to the IDS. DPA, an initial description was presented in [13], however, goes well beyond time machine and FPA as it focuses on the relevant parts of the entire session instead of the first *N* bytes.

The key contributions of this paper are as follows. We performed an in-depth analysis of the rule sets provided for the IDS Snort (Section II). Using this information, we evaluated pattern matches for payload data and examined the rules in regard to overlay protocols and current trends in protocol development. Our findings are supported by live data observed over a period of more than three months. To the best of our knowledge, this has been the first in-depth analysis of the matching position of Snort rules in live traffic (Section III). Our main contribution is the concept of Dialog-based Payload Aggregation. DPA allows filtering and aggregation of payload data belonging to individual TCP connections (Section V). We provide a thorough analysis of DPA's quality of the detection results and its performance improvement by measuring the reduction of data that is achieved in a live network. A performance evaluation of a complete IDS using DPA concludes this part. DPA achieves much better detection rates than approaches such as time machine or FPA as it finds all the relevant parts in TCP session beyond the first *N* bytes.

II. Rule Analysis

We performed an in-depth analysis of current Snort rule sets to get an impression how the rules are structured and what parts of network traffic are relevant for payload-based IDS. We use three sources for the Snort rule sets from September 2009: *a)* Sourcefire (SF),[1] the standard Snort rule set (5625 rules); *b)* Emerging Threats (ET),[2] an open source project (9369 rules); *c)* BotHunter (BH),[3] this project attempts to catch malware communication in networks and uses Snort internally. Some of its rules are taken from ET (2452 rules).

As our work is primarily based on the payload of TCP connections, we removed all rules not matching on TCP data. Our goal is to determine IDS-relevant data portions within the data stream. So we dissected the context of the rules that produced matches in our live tests. In general, rules within the Snort IDS can be differentiated into five usage types: *a) Protocol identification:* Rules designed to identify specific protocols. Often, these rules identify protocols that may violate policies, or serve as preconditions for other application-specific rules. Thus, they do not trigger any events. *b) Application detection:* The primary purpose is to determine specific applications. Often, peculiarities of detected applications are matched, e.g. a specific user-agent string within a HTTP connection. Examples are malware or online games. *c) Exploit detection:* Rules detecting exploits embedded in the data stream. *d) Malware detection:* Corresponding rules detect binary code of malware that is transferred over the network. This binary code is usually transferred either after a successful exploit, or as attachment in other types of data. *e) Protocol state detection:* Rules specialized on detecting state changes within a certain protocol, e.g. failed logins. *f) Usage detection:* General usage identification of network data, e.g. porn detection by searching for typical texts within the data stream regardless of the used protocol.

[1]http://www.snort.org/snort-rules
[2]http://www.emergingthreats.net/
[3]http://www.bothunter.net/

Property	Mean	Minimum	Maximum
Packet rate	55 kpkts/s	14 kpkts/s	130 kpkts/s
Data rate	323 MBit/s	58 MBit/s	790 MBit/s
Detected events	9700	1200	95300

TABLE I
Statistics from 600 10 minute traces used in our evaluation

Internet protocols currently show a trend to multiple layers and nested protocols. We analyzed Snort rules what types of data were searched for by them, and where these data types occurred. Without loss of generality, we go into more detail for HTTP, as it is currently one the most used protocols in the Internet [14] and often serves as overlay protocol for various applications. Data types like XML-RPC or JavaScript are often embedded within HTTP. Within these embedded data types, other data types may be nested, e.g. JavaScript within HTML or HTML within XML-RPC data. As often surrounding protocols add prefix and postfix to the enclosed messages, we can expect, that the "higher" a data type is stacked, the later it will occur within the transport stream. Typical examples in the "Web 2.0" era are social networks with embedded applications, interactive office applications, or geo mapping services.

III. Rule Matching on Network Data

Using live network data, we analyzed the matching positions of the Snort rule sets in more detail. In the following, we outline the test setup and discuss the obtained results.

We used live data from our University's Internet uplink. We captured 10 minute packet traces every 3 hours during the span of 3 months with a packet drop rate of less than 0.01 %. Traffic statistics are shown in Table I. In total, we recorded events from 858 rules, and 526 of these rules produced at least 10 events.

A. False-Positive Events

We observed one source of false positive events within Snort that is often easily avoidable: *loose* rules [15] that not only

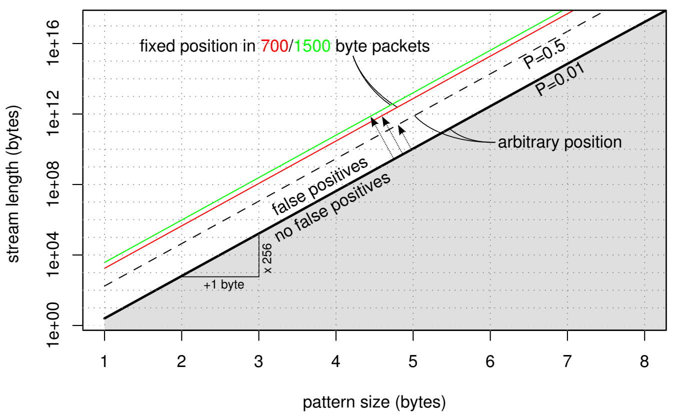


Fig. 1. Probability of false positive pattern matches in random data

match the intended data but data from other traffic as well. These rules sometimes had pattern matches that were too short, and thus generated false positives.

So how many bytes of payload need to be matched to avoid random false positive events for the rule? To answer this question, we performed a statistical analysis of this problem, where we only regarded the pattern size within the payload and ignored any additional filters, such as header filters. Our results are outlined in Figure 1: we compared multiple match cases, a probability $P = 0.01$ and $P = 0.5$ for random false positives and the position of the match in the packet. The position can either be at an arbitrary position in the data stream, also crossing packet boundaries, or fixed at a certain position within a packet. The figure shows the probability for 700 B and 1500 B packets. As an example, a 4 B pattern randomly matches with a probability of $P = 0.01$ in a (random) data stream of 41 MiB. To match with a probability of $P = 0.5$, the data stream needs to be larger by a factor of 69. If the match is fixed to a position within a data stream containing of 700 B packets, 28 GiB of data are required for random false positives with a probability of $P = 0.01$. If the data stream consists of 1500 B packets, 60 GiB would be required. For each additional byte in the pattern, the amount of random traffic needed to obtain the same probability of false positives increases by a factor of 256. Thus, if we consider the data rates of current networks, surprisingly many false positive events will be reported for rules that use a pattern of only 4 B. In practice, some rules alleviate this problem in part by specifying additional filters relying on port or IP ranges.

B. URL matching

A significant amount of rules match content and use Snort's HTTP preprocessor to identify URLs within connection; then only the URLs are parsed for the given string. Usually, these URLs are located at the beginning of HTTP connections and directly after the `GET` or `POST` keyword. Many browsers have optimized the loading process of web pages using *HTTP pipelining*. Thus, a single TCP connection may be used for multiple HTTP requests. Obviously, we cannot assume URLs in HTTP connections to be at the beginning of connections (this idea has been exploited by Time Machine and FPA).

We modified the HTTP preprocessor in Snort to record the position and length of all occurring URLs in our live traffic to determine the usage of HTTP pipelining in current network traffic. We processed network traces with a duration of 60 seconds every 4 hours for several days with the HTTP preprocessor and collected statistics for 1.3 million URLs in 36 traces. In our tests, only around 60 % of all URLs were located at the beginning of TCP connections. So if we only used the beginning of TCP connections for intrusion detection, we would only detect events located in the first HTTP request's URL and risk a significant reduction of detection quality.

IV. FPA

FPA offers a lightweight technique for aggregating a connection's first N bytes of payload in both directions based on the sequence number in the TCP header, or the order of packets in UDP streams [11]. Most of the security-relevant data can be retained this way, but for protocols that exchange control and bulk data within the same connection in an interleaved way, important data may be lost during the aggregation process. DPA, as presented in this paper, extends FPA to collect most of the security relevant events. In order to compared DPA to approaches such as FPA and Time Machine, we briefly analyze the detection quality of FPA.

Figure 2 visualizes both the ratio of analyzed data and detected events with FPA in comparison to all the contained events. The horizontal axis depicts the number of captured bytes from the beginning of a connection that was FPA configured to retain. The upper half of the figure shows an empirical cumulative distribution function (eCDF) of the data ratio selected by FPA to the original amount of data. The lower half visualizes the ratio of detected events by Snort after performing FPA to all events. We removed all false positives that we could reliably identify. Multiple protocols are shown; the protocols have been differentiated by the used ports. Interestingly, all graphs show a rather constant behavior after a FPA length of 1000 B, except for the data ratio of IRC. FPA reduced the amount of data with 2000 B flow length to 1.0%, 1.9%, 7.1%, and 30.0% for HTTP, SSH, SMTP, and IRC, respectively. Although the data reduction is high, Snort was able to detect 83%, 100%, 60%, and 97% of events. The overall traffic was reduced to a portion of 1.0 %, whereby only 78 % of all original events were detected. We expect similar results from the approach used in Time Machine [10] for the same capturing length.

V. New Approach: DPA (Dialog-based Payload Aggregation)

In this section, we introduce a completely new method called Dialog-based Payload Aggregation (DPA). The key objective of this filtering method is to keep the data reduction at a similar level compared to FPA but to significantly increase the detection ratio.

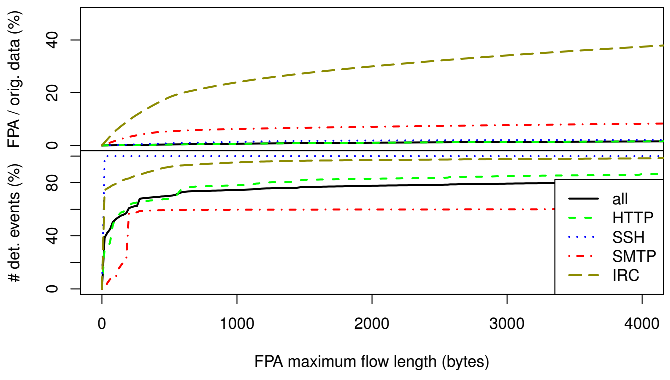
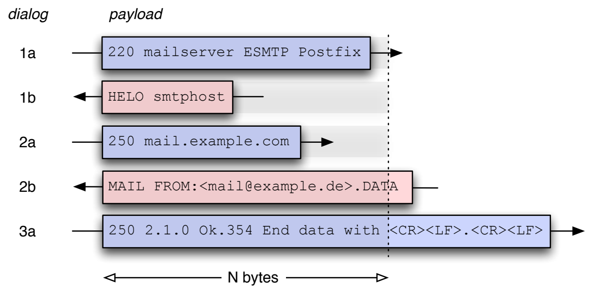


Fig. 2. Data reduction compared to captured events in FPA



dialog
payload
1a
220 mailserver ESMTP Postfix
1b
HELO smtphost
2a
250 mail.example.com
2b
MAIL FROM:<mail@example.de>.DATA
3a
250 2.1.0 Ok.354 End data with <CR><LF>.<CR><LF>
N bytes

Fig. 3. Dialog-based payload aggregation for TCP streams

A. Methodology

DPA operates only on network and transport layer header information to select parts of the transferred payload. The idea is to identify consecutive bidirectional data exchanges in one TCP connection. In the case of HTTP, pipelined requests can be identified by looking at the sequence numbers in the TCP stream. If application data is transferred, the data sender's sequence number will be increased. When the counterpart transfers data, the other sequence number is increased. By monitoring both sequence numbers, a direction change in the communication can be easily detected. The concept of DPA is to select a fixed number of bytes and to send it to the IDS after each direction change. Figure 3 explains the mechanism of our method: a TCP connection consists of several dialog segments. A dialog segment either starts at the beginning of a connection, or after a direction change. It ends when the connection is terminated, or at the next direction change. So, in case of the shown SMTP connection, each request and response are in a separate dialog segment of which the first N byte will be captured. This way, DPA achieves much better results for protocols that mix control and bulk data compared to FPA.

We expect that many IDS rules could be restricted to certain positions within the flow when anchors relative to flow and dialog start were offered by the IDS. Software communicating over network connections often use fixed size buffers for received data. For example, the FTP server software ProFTPD uses in a default configuration 4 KiB buffers. For the previously detected security vulnerability[4], this buffer was used and the corresponding exploit data must have been smaller than 4 KiB, so DPA would have been able to capture the exploit when configured with this size.

B. Dialog Analysis

To our knowledge, dialog segments within TCP connections have not yet been evaluated for intrusion detection. Thus, we first analyzed their general properties for different application protocols. Figure 4 (left) shows an eCDF of the length of dialog segments in the observed TCP connections. We evaluated both directions of a TCP connection separately. Using port filters, we differentiated protocols HTTP, SMTP and SSH, but also depicted the total network traffic ("all"). All HTTP dialog segments that were transferred to the server show the expected behavior for HTTP requests: 80 % lie in the range of 300 B to 1500 B. The responses either contain an error message, whose size is often around 300 B, or bulk data with a much higher segment size. Protocols using several rounds of bidirectional communication, e.g. SMTP or IMAP, show a much lower average segment size. Here, many small requests are sent by the client, where most of them trigger short responses like status messages from the server. Only a small percentage of less than 5 % of the dialog segments is larger than 2 kB. We also included the encrypted protocol SSH in the figure. Clearly, the client-side is often the interactive part, as its median dialog segment size is 65 B, whereas the median of dialog segments from the server is 110 B. Server and client traffic for SSH shows multiple spikes, for example at around 300 B and 520 B. We were able to trace this back to several scans and brute force cracking attempts that frequently repeated similar protocol exchanges. In conclusion, we would be able to capture more than 78 % and 94 % of whole dialog segments to the server and client, respectively, using DPA to record a maximum dialog segment size of 2 kB.

We also analyzed the number of dialog segments in a connection. Figure 4 (right) shows an eCDF of the number of dialog segments. Here, we ignored the transfer direction. In HTTP, more than 80 % of all connections contained exactly two dialog segments – one for the HTTP request, one for the HTTP response from the server. All connections with more dialog segments use pipelining, and most connections contain a multiple of 2 dialog segments. Protocols, where the server sends a greeting message before the client issues a command, e.g. SMTP or FTP, show a strong preference to an uneven number of dialog segments. In the case of SMTP, 3 to 7 dialog segments are commonly used. Although mostly the same commands were issued, sometimes multiple commands were sent in a row, so that individual requests were grouped in one dialog segment. This is not the case for SSH, as almost every command issued for establishing the secure connection depends on the response of the previous command. In the figure, only a few connections contained between 2 to 9 dialog segments. The spike at 10 dialog segments was caused by the

[4]Bugtraq ID 44562, published on 2010-11-01

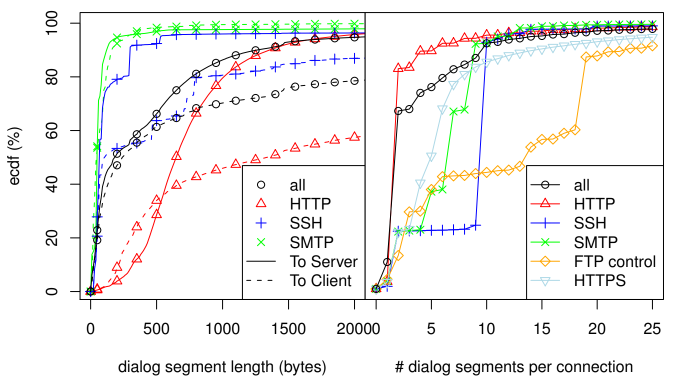


Fig. 4. Left: Length of dialog segments; Right: Number of dialog segments in TCP connections

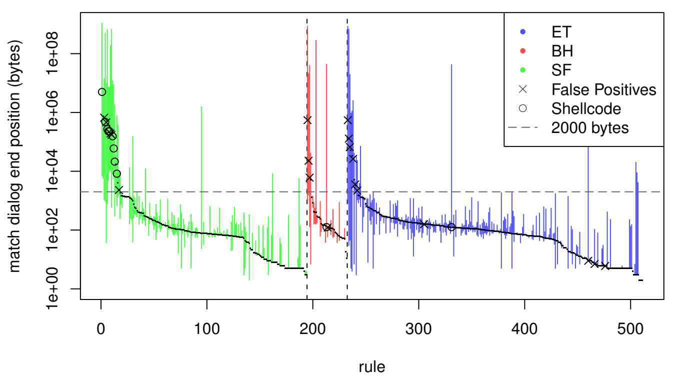


Fig. 5. End of matches in live data relative to the start of the dialog segment

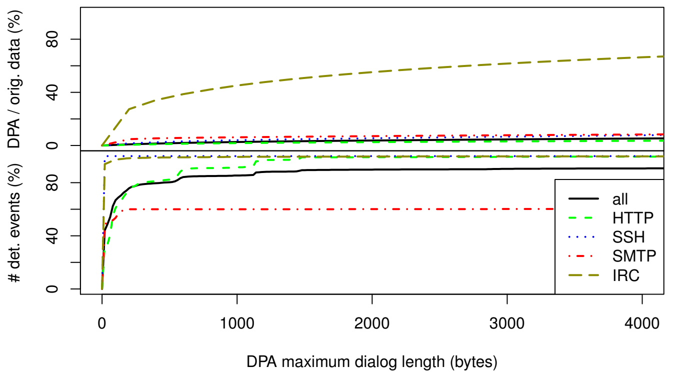


Fig. 6. Data reduction compared to captured events in DPA

already mentioned SSH scan. FTP is a rather lengthy protocol in bidirectional communication, as for one download multiple commands need to be issued. A typical value for current FTP clients is 19 dialog segments, which confirms a spike at this position in the dialog segment count in Figure 4 (right).

C. Detection Quality

In order to estimate the event detection quality using DPA, we recorded the end position of matches in our live network data. Figure 5 shows the results for the position relative to the dialog segment start. A point on the horizontal axis marks a single rule and the logarithmic vertical axis shows the end of matches relative to the start of the dialog segment. All matches found for a rule are summarized in a vertical line that marks the 5 % and 95 % quantile of the matches. The median is shown by a black dot, cross or circle, and the line color depicts the type of rule set the rule originated from. The majority of rules have medians clearly within the 2000 B boundary. With FPA, 52 rules violated this boundary, whereas this number was reduced to 31 with DPA. Interestingly, the median of some rules that we identified to produce false positive events was highly reduced. A few true-positive events also showed a high reduction of the match position (e.g., URI matches in pipelined HTTP connections). Many other true-positive events either showed a slight decrease in the match position (e.g., SMTP buffer overflow matches) or none at all. After removing shellcode and false-positive events, we observed only 8 rules that generated a high amount of matches after the 2000 B boundary. All of them show unusual protocol behavior, as their rule semantics require them to be located at the front of a dialog segment, but their match position was later in the segment. We strongly suspect these matches to be false-positives, but we did not mark them as such because of ambiguities caused by trace anonymization. If we removed all identified shellcode rules, we would capture up to 95 % of all events with a maximum dialog segment size of 2000 B.

Figure 6 visualizes both the ratio of analyzed data and detected events with DPA in comparison to the analysis without DPA (the same experiment has been described in Figure 2 for FPA). The graphs show quite constant behavior after a capture length of 500 B. DPA reduced the amount of data with 2000 B capture length to 2.5%, 5.4%, 7.0%, and 55.2%, for the protocols HTTP, SSH, SMTP, and IRC, respectively. Compared to FPA, the amount of data is $2-3$ times higher in average. For the given protocols, Snort was able to detect 99.2%, 100%, 60.1%, and 99.7% of events. The low detection rate for SMTP is caused by shellcode matches that were predominantly contained inside mails. The overall traffic was reduced to a portion of 3.7 %, thus achieving theoretical IDS performance speed-up of more than 25 times assuming a linear relationship between data rate and processing costs, while still detecting 89.8 % of all original events. Please note, that rules producing a high number of events have a high influence on the aggregate value in this figure.

D. Performance Evaluation

So far, we analyzed the data reduction as achieved by DPA and estimated the theoretical speed-up. As there will be not exactly a linear relation between data rate and computational costs, we measured the performance gain in our testbed using our monitoring framework Vermont in combination with Snort as illustrated in Figure 7. We used two separate machines, a traffic generator sending previously recorded network traffic, and a monitoring system.[5]. Each test run consists of a 10 min

[5]Intel Core2 Quad CPU @ 2.83 GHz; Intel 82572EI Ethernet controller

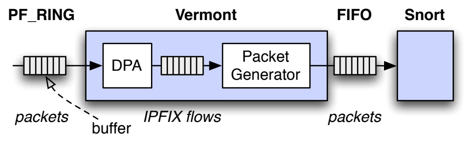


Fig. 7. Test setup used for the performance evaluation

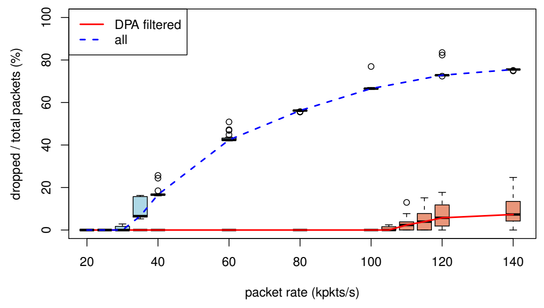


Fig. 8. Dropped packet ratio in 30 test runs with and without DPA

packet trace matching the given packet rate. All packets are captured by an performance-improved version of the PCAP library called PF_RING [16]. Flow elements consisted of one or two dialog segments of up to 2000 B. After flow aggregation, we used a buffer of 100 000 elements before converting each incoming flow record to one (or two) packets, according to the payload contained in the DPA fields. Then these packets were forwarded to Snort using a pipe stream for inter-process communication, and Snort was configured to read packet data from standard input. To compare the results with a standard system, we also performed an evaluation using Snort that directly captured packets from PF_RING without DPA.

Figure 8 shows the relative amount of dropped packets for different packet rates. The blue dashed line shows Snort directly analyzing all incoming packets, the red continuous line shows the test setup using Vermont, where only DPA-filtered traffic is forwarded to Snort. We evaluated packet rates of $20 - 140$ kpkts/s (140 kpkts/s transferred roughly 920 Mbit/s). Each experiment was executed 30 times. We plotted the results in form of a boxplot. The lines connecting the boxplots connect the median of each test configuration. Snort shows first packet drops at 30 kpkts/s, whereas Snort in combination with DPA produced no packet losses at packet rates up to 100 kpkts/s.

Our test implementation is not optimal regarding performance, as some processing stages were performed twice, such as stateful connection tracking with TCP reassembly in Vermont and Snort. Furthermore, the buffer between Vermont and Snort only held 4 KiB, which is too small for our test data rates. We expect that by removing these deficiencies we would achieve even higher speed-ups than 300 %.

VI. Conclusion

We performed a thorough analysis of payload filtering methods that are suited for network-based IDS. Using extensive traces from a live network data and three different Snort rule sets, we evaluated detection results applying FPA to the input data. Based on our findings, we suggest a new aggregation method, Dialog-based Payload Aggregation, that exploits intrinsic application protocol semantics to extract security-relevant portions of the payload for subsequent analysis. This method is very lightweight, as the decision process only uses network and transport header data from captured packets. According to experiments using our traces, we could filter out 96 % of all network data, while retaining 89 % of all events reported by Snort.

References

[1] M. Roesch, "Snort: Lightweight Intrusion Detection for Networks," in *13th USENIX Conference on System Administration (LISA 1999)*, Seattle, WA, November 1999, pp. 229–238.

[2] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Elsevier Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, December 1999.

[3] M. Handley, V. Paxson, and C. Kreibich, "Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics," in *10th USENIX Security Symposium*, Washington, DC, August 2001.

[4] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational experiences with high-volume network intrusion detection," in *11th ACM Conference on Computer and Communications Security (ACM CCS 2004)*. Washington, DC: ACM, October 2004, pp. 2–11.

[5] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," IETF, RFC 5101, January 2008.

[6] K. Namjoshi and G. Narlikar, "Robust and Fast Pattern Matching for Intrusion Detection," in *29th IEEE Conference on Computer Communications (INFOCOM 2010)*. San Diego, CA: IEEE, March 2010.

[7] R. Smith, C. Estan, S. Jha, and S. Kong, "Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata," in *ACM SIGCOMM 2008*, Seattle, WA, August 2008, pp. 207–218.

[8] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. P. Markatos, and S. Ioannidis, "Regular Expression Matching on Graphics Hardware for Intrusion Detection," in *12th International Symposium on Recent Advances in Intrusion Detection (RAID 2009)*, vol. 5758, Saint-Malo, France, September 2009.

[9] P. Tune and D. Veitch, "Towards Optimal Sampling for Flow Size Estimation," in *8th ACM SIGCOMM Conference on Internet Measurement (IMC 2008)*. Vouliagmeni, Greece: ACM, October 2008, pp. 243–256.

[10] S. Kornexl, V. Paxson, H. Dreger, R. Sommer, and A. Feldmann, "Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic," in *5th ACM SIGCOMM Conference on Internet Measurement (IMC 2005)*. Berkeley, CA: ACM, October 2005, pp. 267–272.

[11] T. Limmer and F. Dressler, "Flow-based Front Payload Aggregation," in *34th IEEE Conference on Local Computer Networks (LCN 2009): 4th IEEE LCN Workshop on Network Measurements (WNM 2009)*. Zurich, Switzerland: IEEE, October 2009, pp. 1102–1109.

[12] R. T. Lampert, C. Sommer, G. Münz, and F. Dressler, "Vermont - A Versatile Monitoring Toolkit for IPFIX and PSAMP," in *IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006)*. Tübingen, Germany: IEEE, September 2006, pp. 62–65.

[13] T. Limmer and F. Dressler, "Dialog-based Payload Aggregation for Intrusion Detection," in *17th ACM Conference on Computer and Communications Security (CCS 2010), Poster Session*. Chicago, IL: ACM, October 2010, pp. 708–710.

[14] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On Dominant Characteristics of Residential Broadband Internet Traffic," in *9th ACM SIGCOMM Conference on Internet Measurement (IMC 2009)*. Chicago, IL: ACM, November 2009, pp. 90–102.

[15] R. Sommer and V. Paxson, "Enhancing byte-level network intrusion detection signatures with context," in *10th ACM Conference on Computer and Communications Security (ACM CCS 2003)*. Washington, DC: ACM, October 2003, pp. 262–271.

[16] L. Deri, "Improving passive packet capture: beyond device polling," in *4th International System Administration and Network Engineering Conference (SANE 2004)*, Amsterdam, The Netherlands, September 2004.