The 4th IEEE LCN Workshop on Network Measurements (WNM 2009)
Zürich, Switzerland; 20-23 October 2009

Flow-based Front Payload Aggregation

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

***Abstract*—We present and discuss a new monitoring technique that we call Front Payload Aggregation (FPA). Instead of being limited to either analyzing single packets for signature-based attack detection or exploiting statistical flow information for anomaly detection, FPA combines the advantages of both approaches. Exploiting the fact that most attack signatures can be found in the very first packets of a connection, we collect payload information from these few packets (we take the first n payload Bytes) and associate it to the corresponding flow data. Thus, intrusion detection can still be performed with a high degree of confidence and the monitoring system becomes efficient w.r.t. processing performance and attack resilience.**

I. Introduction

State-of-the art network-based Intrusion Detection Systems (IDSs) like Bro [1] or Snort [2] analyze each received packet including all contained payload information in order to detect ongoing attacks. Additionally, these systems offer high resilience against targeted spoofing attacks by performing validity checks on all packets whether these may have been accepted by the receiving endpoint of the connection. They offer high quality detection properties, however, due to their computationally intensive analysis methods, they do not cope well with current network speeds. Looking at the trend of network speeds in the future, this situation will not work out for these systems. So in-depth analysis is only possible with fast and expensive hardware.

The alternative to in-depth IDSs are anomaly detection algorithms, which are often based on aggregated data from packet headers, so-called flows. Algorithms range from portscan detection [3] to anomaly detection in data rate usage [4]. As their input data is usually very coarse, it is often easy for attackers to circumvent detection. A trade-off is needed, that combines both approaches and still offers good detection ratio at fast network speeds.

Recent publications indicate, that most of the security-relevant data is transferred in the beginning of a connection, i.e. the first n bytes of each connection [5]–[7]: in this part of the connection, the data transfer is initialized. Both client and server perform a handshake on the application layer and exchange the data request and response. After this initial exchange, often a bulk of data is transferred. Take HTTP for an example: the important information exchange would be represented by the HTTP GET or POST request that includes the client version, the location of the requested data, client capabilities, possibly username and password, and so on. The same applies to the HTTP response from the server.[1] Many exploits for web applications transfer their exploit code within a short HTTP request, like in the given Uniform Resource Locator (URL). Other examples for security-relevant network data is created by malware communication using IRC or HTTP as application protocol or the SMTP protocol used for sending spam. This way, the heavy-tailed nature of today's network traffic [8] can be exploited to reduce the amount of data to be analyzed by ignoring the largest part of long flows.

[1] See also our example in Figure 2, where a HTTP connection's first 256 bytes of payload are shown

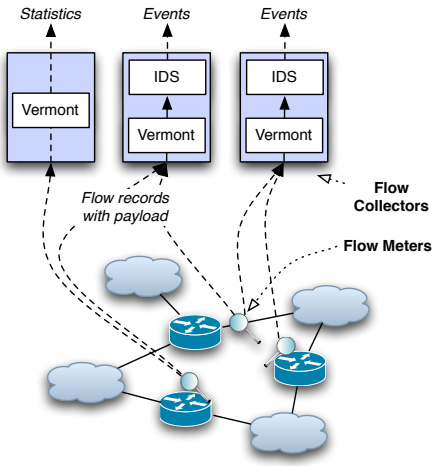


Fig. 1. Example attack detection system with multiple sensors and IDS

Flow aggregation has long been present in the area of network monitoring, with an emphasis on traffic accounting and performance measurement. The standardized Internet Protocol Flow Information Export (IPFIX) protocol is already supported by a wide range of hardware-based routers and software monitoring solutions running on commodity PC hardware [9], [10]. Our approach is to merge the concepts of flow aggregation and payload-based IDS. We call this technique Front Payload Aggregation (FPA), where the first n payload bytes of a flow are added to the flow records and, consequently, this data is directly included into the IPFIX data stream. We propose FPA, a fast reassembly algorithm that uses a constant buffer size and does not require additional memory during the aggregation process.

An exemplary attack detection system is depicted in Figure 1. Multiple flow meters capture data at various network links. These include statistical data about monitored flows, as well as parts of the payload. Flow collectors receive these flow records, and may perform statistical analysis and anomaly detection on the aggregated data, and other collector instances may extract

payload from the flow records and forward it to off-the-shelf IDSs. Conventional approaches often involve the deployment of an IDS at a central network link in a static setup, where only data transferred on this single link can be monitored by the IDS. By separating the IDSs from the network link, an additional layer is introduced that enables the system to adapt itself dynamically to current load or other external triggers like handling anomalies: if one host within the local network becomes conspicuous, it will be possible to monitor data that is not transferred over the centrally monitored link, but individual flow meters within the network may be triggered to selectively report data from the conspicuous host.

The contributions of this paper can be summarized as follows. We discuss the need for processing at least the first n payload byte in each connection for efficient signature-based attack detection. As current flow monitoring techniques abstract from any payload and only provide statistical information, we introduce a new flow-monitoring technique, Front Payload Aggregation, which we defined as an extension to the standardized IPFIX protocol (Section III). According to the performance evaluation results, FPA only requires marginal overhead (CPU, memory) compared to normal flow processing (Section IV). In conclusion, we show that FPA provides capabilities for high-quality signature-based attack detection while keeping the resource requirements at an adequate level comparable to normal flow monitoring.

II. Preliminarities

A. Related Work

A considerable amount of work has already been published focusing on the analysis of payload-based IDSs that perform application layer analysis for selected protocols and signature matching on pre-defined rules. The best known examples are Bro [1] and Snort [2]. Experiences in high-speed networks with Bro including information about the processing performance and the memory utilization have been presented in [11]. Furthermore, the feasibility of performance prediction of some Bro preprocessors has been described in [12]. Further work has been conducted to improve Bro's performance by running the system on multiple systems in parallel [13]. Similarly, the system performance has been doubled by offloading the signature matching to graphic cards [14]. Recently, a system has been introduced that saves a network link's history using flow records with added payload [15]. In this system, each connection's payload is cut off after 10-20 kBytes to save space.

Efficient TCP stream reassembly is covered in [16]. As passive TCP reassembly does not resolve all ambiguities in the traffic, a form of traffic normalization to remove any effects of attack has been proposed [17]. Finally, both approaches have been combined by introducing a TCP reassembly system that offers robustness against attacks by modifying monitored traffic inline [18].

The rule sets of the IDS Snort have been analyzed and evaluated in order to determine how much payload within a packet is needed for effective attack detection [7]. The authors analyzed Snort rules and determined the minimum amount of payload for each rule so that it may match successfully.

B. Flow Monitoring

Flows are sets of IP packets sharing common properties. A flow record contains information about a specific flow. In most applications, a typical configuration would be using the IP 5-tuple *<source IP, dest IP, source port, dest port, protocol>* as flow keys, i.e. attributes describing the flow. Furthermore, relevant statistical data can be added such as the flow start and end times or the number of bytes of all packets belonging to the flow. Several protocols are available to efficiently transfer flow records. Most of the state-of-the-art flow meters support either Netflow.v9 or IPFIX. The latter one was standardized by the IETF in RFC 5101 [19]. Both protocols support variable configurations: so called template records that describe the structure and content of flow records, are transmitted regularly according to predefined timeouts. For flow aggregation, the active timeout describes the maximum time a flow record is kept in cache and the passive timeout is used if no more packets are received for the particular flow. Packet Sampling (PSAMP) [20] is a derivative of IPFIX that also allows the inclusion of packet payload. Each received packet then is represented by one flow record. The protocol specification also emphasizes on various sampling methods for the selection of packets.

C. Vermont

For our live performance tests with FPA, we used Vermont [9]. It is an open-source monitoring toolkit capable of processing Netflow.v9 and IPFIX conforming flow data. The application runs on Linux and derivatives of BSD. Vermont can receive and process raw packets via the Packet Capturing (PCAP) library (up to 1 GBit/s) as well as IPFIX/Netflow.v9 flow data. Supported data formats for export are IPFIX, PSAMP, and the Intrusion Detection Message Exchange Format (IDMEF).

Internally, Vermont is heavily multi-threaded. Its design allows to define modules with different functionalities in the configuration and to connect them either directly or using internal queues. For our evaluation tests, Vermont is well suited, as tasks like reading packets from the network via PCAP, flow aggregation and flow export are realized in separate threads. A sensor framework within Vermont supports detailed analysis of each module's performance attributes during run-time.

III. Methodology

In the following, we outline the main concept of FPA and study the working behavior of the algorithm. Performance aspects are discussed in Section IV.

A. Goals

The following design goals led the design of our FPA algorithm:

a) High performance: The history of Netflow and IPFIX metering systems lies in area of traffic accounting, which focuses on statistical analysis of network traffic. So these systems have been designed to enable aggregation of data packets in high speed networks, and to touch as few bytes as possible within each packet. Normal 5-tuple flow aggregation only accesses the network and transport layer headers within the packets. Thus, only 40 Byte have to be processed (IP options are usually not accessed). In contrast to attack-resilient IDS like Bro [16], no data validation is performed such as checksum verification or stream analysis, in which acknowledgments in TCP connections are verified.

b) Simplicity: One of the major advantages of Netflow and IPFIX is its wide availability in both software and hardware based systems. This is caused by the simplicity of its design. The same objective holds for FPA.

c) Seamless integration into current flow processing systems: IPFIX can easily be extended by developers using enterprise-specific (i.e. user-defined) fields. Unknown fields are usually ignored by flow processing systems, if data is stored or forwarded. This way it is possible to integrate FPA without changing the flow processing chain.

B. Aggregation Method

FPA is performed using two new enterprise-specific information elements in the IPFIX flow (to be specified in the template): `IPFIX_ETYPEID_frontPayload` (f_{buf}) represents the buffer for the aggregated payload and has a variable size that is to be specified in the template. `IPFIX_ETYPEID_frontPayloadLen` (f_{buflen}) contains the actual length of payload within the buffer using a 32 Bit unsigned integer. We restricted aggregation to the protocols UDP and TCP, as these protocols transfer most of the payload relevant for security in today's networks. However, our FPA technique can easily be extended to support other transport protocols such as SCTP. An example flow record with included FPA buffer is displayed in Figure 2.

TCP packets contain a sequence number p_{seq} to indicate the packet's position within the TCP stream. According to RFC 793 [21], a random number is chosen by both communicating partners during connection establishment. After the initial SYN handshake, the sequence number indicates the position within the TCP stream of the payload contained in the packet. This "'position marker"' can easily be exploited for payload aggregation.

```
SrcIP: 1.2.3.4
DstIP: 5.6.7.8
SrcPort: 1558
DstPort: 80
FlowStart: 2009-04-04 22:54:27.432
FlowEnd: 2009-04-04 22:54:31.252
Octets: 904
RevOctets: 13830
Packets: 11
RevPackets: 12
Protocol: 6
PayloadLen: 256
RevPayloadLen: 256
Payload: GET /pub/mozilla.org/addons/3006/video_downloadhel
      per-4.2-fx.xpi HTTP/1.1..Host: releases.mozilla.or
RevPayload: HTTP/1.1 200 OK..Connection: close..Transfer-En
      coding: chunked..Date: Thu, 04 Apr 2009 22:54:28 GMT.
```

Fig. 2. Content of a flow record with activated FPA (non-printable characters removed from payload)

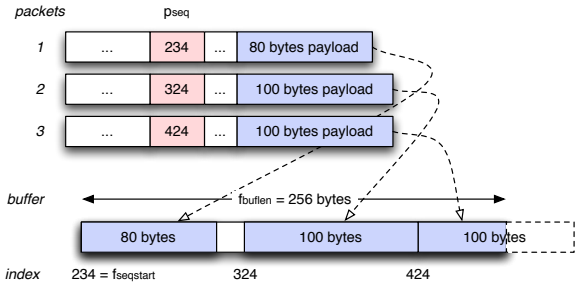


Fig. 3. Front payload aggregation for TCP packets

Algorithm 1 and Figure 3 explain the principles of our new FPA method. Each flow record contains a buffer with fixed size f_{buflen}. If a new flow is being monitored, i.e. if a packet arrives for which no flow information has already been recorded, the internal variable $f_{seqstart}$ is set to the packet's TCP sequence number and marks the buffer's start index in the TCP stream. f_{bufuse} is set to 0 to indicate that the buffer is not filled with any data. If the SYN flag is set within the packet, it is part of a TCP connection handshake and does not contain any payload. Then $f_{seqstart}$ is increased by 1 as the following packet would contain $p_{seq} = f_{seqstart} + 1$ (see [21]). For each incoming packet, it is tested whether

- the packet contains payload,
- the sequence number of the packet p_{seq} lies in range $[f_{seqstart}, f_{seqstart} + f_{buflen})$, or
- the number of packets in the recorded flow f_{pkts} meets $f_{pkts} < 2.8 \times 10^6$.

If all tests are positive, the packet's payload is copied to the buffer (but cropped if needed). Usually, only the payload of the first few packets of a flow is copied into the buffer. Afterwards, the range test fails when the sequence number increases. The

Algorithm 1 Front payload aggregation for TCP packets

```
1: receive TCP packet p
2: extract p_seq and p_len
3: if first packet in flow then
4:     set f_seqstart = p_seq
5:     if SYN-bit is set in packet then
6:         set f_seqstart = f_seqstart + 1
7:     end if
8:     set f_bufuse = 0
9: end if
10: if f_pkts < 2.8 × 10^6 and TCP payload is in packet and p_seq
    indicates that payload fits in buffer then
11:     copy payload in buffer
12:     adjust f_bufuse
13: end if
```

third test is performed as TCP sequence numbers are 32 Bit unsigned integers and wrap after a transferred payload of 2^{32} Byte. This is the case after at least $2^{32}/1500 \approx 2.86 \times 10^6$ packets in Ethernet with a Maximum Transmission Unit (MTU) of 1500 Byte.

No stream reassembly is possible for UDP packets, because no sequence numbers are available in the protocol. So, the payload of all incoming packets for a flow is copied consecutively into the FPA buffer until it is filled. $f_{seqstart}$ is not used at all, and f_{bufuse} is increased per packet. Algorithm 2 illustrates the procedure.

C. Discussion

This section provides an analysis of the FPA algorithm described previously, both in handling anomalous traffic and security issues. The first part discusses general properties of FPA. After that, both algorithms are analyzed in detail.

Standard flow aggregation settings that only regard the headers of the network and the transport layer need to access the first 40–120 Byte of a packet (excluding MAC layer information and depending on the header sizes of both layers). If payload is needed for flow aggregation, it is obvious that more data from each packet needs to be captured. As the whole FPA buffer size f_{maxlen} may be contained in the payload of a single packet, the capture length should be the expected maximum header size plus f_{maxlen}, i.e. $120 + f_{maxlen}$. In practice, the maximum size of 60 Byte per header is not reached. Thus, usually varying capture lengths between 64 and 80 Byte are used for capturing only network and transport layer headers.

Flow records without FPA and standard fields often have a size of 50 Byte. FPA increases storage requirements and transfer data rates immensely: for moderate buffer sizes like 128 Byte, flow record sizes are increased by 156 %. Especially during Denial of Service (DoS) attacks that involve a huge amount of packets, each flow record contains spoofed source data and no payload at all, and the increased flow record size may cause problems. In extreme cases, it is possible that more traffic is generated with flow records than was monitored. This is always the case if flows are monitored that consist of fewer byte compared to a flow record. In our case of 178 Byte flow records, flows consisting of single packets with less than 178 Byte generate more data in the monitoring system than on the network link. In order to generate $d_{mon} = 100\,\text{Mbit/s}$ traffic by the monitoring system configured with 128 Byte FPA buffers (i.e., a flow record size of $f_{size} = 178\,\text{Byte}$), the following packet rate r_{pkt} and data rate r_{bit} of packets with size $p_{size} = 40\,\text{Byte} = 320\,\text{Bit}$ would be needed to sent:

$$r_{pkt} = \frac{d_{mon}}{f_{size}} \approx 73\,600\,\text{packets/s} \quad (1)$$

$$r_{bit} = \frac{d_{mon}}{f_{size}} p_{size} \approx 22.4\,\text{Mbit/s} \quad (2)$$

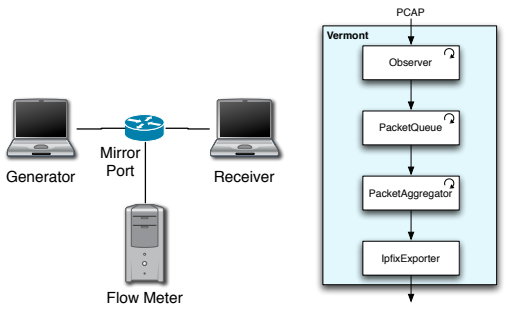
Here the data rate generated by flow records is more than 5 times higher than the data rate of the monitored link. Flow aggregation is performed to reduce monitoring information, and this goal is not achieved in these extreme cases. Additionally, DoS attacks generating high amounts of flow records within the aggregation buffer also often tend to overload the flow meter even without FPA. With normal flow record sizes of 50 Byte, traffic amplification is still possible, but not as high as with activated FPA. Some approaches already try to cope with this problem by installing measures to detect DoS attacks and filter traffic belonging to them before it reaches the aggregation buffer.

FPA of TCP packets performs simple stream reassembly. Due to timeouts within the aggregation, it is possible that a flow record is exported although its corresponding TCP connection has not ended yet. If such a late packet has been received, the flow meter has no way of telling if the connection has already been monitored before. Thus, the connection is considered new and a new flow record is created with its payload filled into the FPA buffer. Due to the simplistic nature of FPA, anomalies like packet retransmissions, bit flips in packets, or packet drops are handled consistently, but not always optimal in the sense of generating flows with correct payload, i.e. payload that was received and interpreted by the receiving end of the connection (in contrast to being dropped). Packet retransmissions in TCP are no problem for FPA, because if the packet's p_{seq} fits in the FPA buffer, the packet's payload is copied into the buffer again. Packet drops often introduce packet flows with non-monotonic increasing sequence numbers when the lost packet is retransmitted. FPA also handles these cases in a similar way. Packets with corrupted payload data, i.e. such with an invalid checksum, may cause problems to FPA. As the chronologically last packet fitting in a buffer with correct p_{seq} always overwrites data in the FPA buffer, invalidated payload may be copied into the buffer. However, because TCP usually retransmits packets with invalid checksums and these retransmissions are monitored after receiving the invalid packet, the FPA buffer is automatically corrected. This behavior cannot be ensured because there is a small chance that, due to packet reorderings within the network, invalid packets may be monitored and aggregated last.

Payload aggregation of UDP packets has the same problems as reassembly of UDP payload data. UDP does not offer a way of knowing the correct order of packets without the help of the application layer protocol. So, it is not possible at all to detect packet retransmissions, reorderings, or losses. The FPA buffer contains the payload of consecutively monitored packets without any reassembly logic at all.

Algorithm 2 Front payload aggregation for UDP packets

```
receive UDP packet p
extract p_len
if first packet in flow then
    set f_bufuse = 0
end if
if UDP payload is in packet and f_bufuse < f_buflen then
    copy payload in buffer at position f_bufuse
    adjust f_bufuse
end if
```



(a) Lab setup (b) Vermont module configuration

Fig. 4. Test setup used for the performance measurements

Even for IDS using state-of-the-art full packet analysis with checksum and acknowledgment checking, it is not possible to unambiguously determine which monitored packets are accepted by a receiver. Attackers are able to exploit this problem when directly targeting the monitoring system and intentionally create ambiguities in the packet stream. FPA does not circumvent this problem with its simple reassembly algorithm, so attacks directly targeted at the aggregation method cannot be detected. So for example, attackers may be able to fool the system by repeating packets with identical *pseq*, but different payload and invalid checksum. Then, the receiving host drops the packet whereas FPA copies the payload data into the buffer and an attack may be masked.

IV. Evaluation

A. Test setup

Our primary goal for the evaluation of FPA was to get realistic results. Therefore, we set up multiple systems to perform direct packet capture from a mirrored port. The test setup for our implementation of FPA within Vermont is depicted in Figure 4(a). One system sends packets, which were read from a previously generated trace file using *tcpreplay*,[2] to a receiving host. All generated network packets are mirrored by a switch to a host running Vermont, which finally captured all incoming packets.[3]

Figure 4(b) shows Vermont's configuration during the test, whereby open circle arrows indicate autonomous threads within modules. Packets are captured using a memory-mapped version of the PCAP library[4] by a thread inside the Observer module. These packets are forwarded to a queue that buffers the received elements. The queue's thread inserts and aggregates the packets into a hash table inside module PacketAggregator. The thread in the PacketAggregator module exports the resulting flow records to the IpfixExporter module in intervals of 10 s. Passive flows are exported after 35 s and active flows after 130 s. We configured three threads on purpose to be able to measure separate tasks inside the flow meter individually: the Observer thread reports the time needed to read packets from the network via PCAP and build a simple wrapper around the raw packet. The PacketQueue thread inserts incoming packets into the hash table and performs data aggregation. Finally, the PacketAggregator thread checks the hash table regularly for expired records and forwards these to IpfixExporter.

Normal flow aggregation only accesses very few bytes within the headers of each monitored packet. FPA increases the number of accessed bytes dramatically for selected packets, as payload is copied from individual packets. Thus, an important indicator for performance is the number of packets within each flow where payload was copied from the packet to the flow record. We recorded this value for selected tests.

Four our evaluation tests, we used a trace that was directly captured from the Internet uplink of our University's network. It has a length of 5 min and was replayed into the test network at the original speed.

[2]http://tcpreplay.synfin.net/trac/

[3]The monitoring system was running on an Intel Core 2 Quad Q9550 at 2.83 GHz, the packets were captured with an Intel 82566DM-2 network interface.

[4]http://public.lanl.gov/cpw/

TABLE I
Traces used for the evaluation

	filter	packet rate	bit rate
L_{ALL}	–	61 kpkt/s	337 Mbit/s
L_{HTTP}	port 80	32 kpkt/s	206 Mbit/s
L_{SSH}	port 22	1 kpkt/s	6 Mbit/s
L_{SMTP}	port 25	1 kpkt/s	3.3 Mbit/s
L_{IRC}	port 6667	0.08 kpkt/s	0.08 Mbit/s

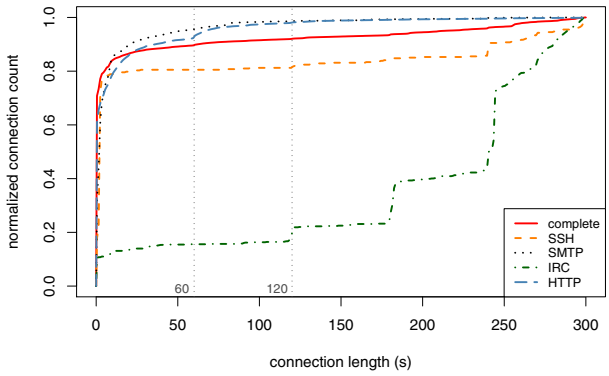


Fig. 5. Cumulative connection length distribution of live dumps

B. Generator traces

The live network trace shows a high variety of different traffic types: our University's network supports multiple high-profile servers, a multitude of workstations, and privately used hosts in dormitories. The trace was captured using *tcpdump* with a maximum capture length of 1500 Byte per packet on 3/4/2009. This setting resulted in about 0.005 % packet loss. To show the effects of different traffic types, we modified this trace for the protocols HTTP, SSH, SMTP, and IRC by

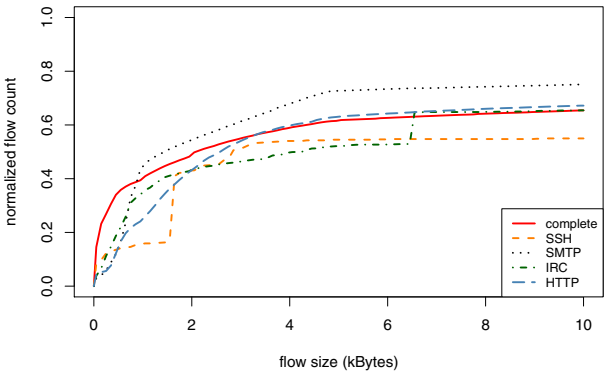


Fig. 6. Cumulative flow size distribution of live dumps

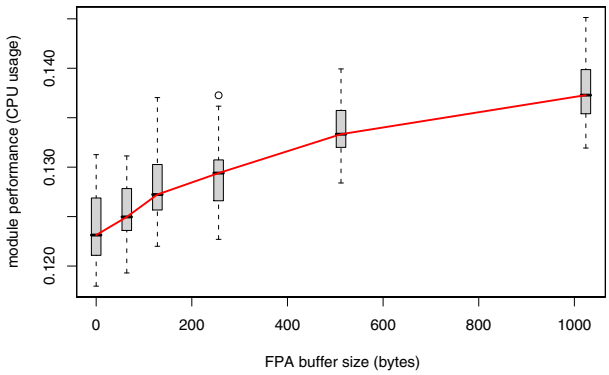


Fig. 7. Module performance for module observer, differing FPA lengths

filtering the original trace according to the ports 80, 22, 25, and 6667, respectively. A summary of the trace statistics is shown in Table I. In order to provide a more detailed overview of our live traces, we also show the distribution of the connection lengths in Figure 5, as well as the distribution of each connection's total size in Figure 6, where the size of the MAC and the network layer headers is included. We only included the size range between 0–10 kByte, as this is the important range for FPA buffer sizes $\leq$1024 Byte. All live dumps except L_{IRC} show a trend for connection lengths below 30 s. Therefore, most of the connections will be correctly aggregated to single flow records without any splits caused by short buffer timeouts inside the flow aggregator.

C. Results

In many of our graphs, we use so-called boxplots. For each set of test results using the same parameters, a box is drawn from the first quartile to the third quartile, and the median is marked with a thick line. Additional whiskers extend from the edges of the box towards the minimum and maximum of the data set. The red line in the graphs connects the medians of the data sets.

1) PCAP: We use the memory-mapped PCAP library in all our tests. This modified PCAP library offers two features for high-performance monitoring in the Linux operating system: a ring buffer with a variable size where the kernel stores all packets captured at the network interface; furthermore, it removes the need for a second copy operation of the packet contents before it reaches the application. The ring buffer is set in our tests to a size of 64 000 frames, whereby each frame may hold one packet. Even during performance glitches of the application, packets are not lost at the interface as long as the ring-buffer can cache the incoming packets.

Our modularized flow meter Vermont is very well suited for measuring the performance of this PCAP library, because the Vermont module Observer, that directly interfaces with it, receives the packets and performs one additional copy operation on the packet data. This copy operation is needed, because the

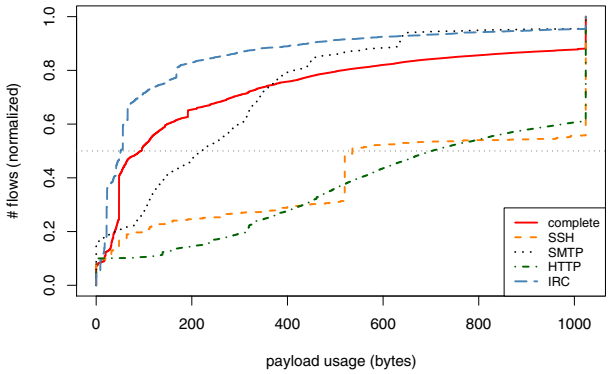


Fig. 8. FPA buffer usage in live dumps

PCAP ring buffer does not guarantee that after a new packet is delivered to the application, content of the previous packet is still available. Vermont's modularized structure does not pose limitations on how long one packet is available, so Vermont needs to guarantee the availability by copying each packet.

For normal flow aggregation, only the packet headers up to the transport layer need to be captured. Thus, usually the PCAP capture length is set to 80 Byte or 96 Byte. To show the effects of increased capture lengths on PCAP, we fed our complete live dump L_1 to Vermont and measured the performance of module Observer. Figure 7 shows the results: For each FPA buffer size s, the PCAP capture length is set to $s+80$. With a standard capture length of 80 Byte without FPA, the module has an average CPU utilization of about 12.4 % (average value measured per second). The maximum capture length was set to 1104 Byte and the average CPU utilization of 13.8 % shows that increased capture lengths only marginally affect the PCAP performance in this setup.

2) FPA buffer usage: Flow meters usually have a static configuration that specifies aggregation rules and the layout of

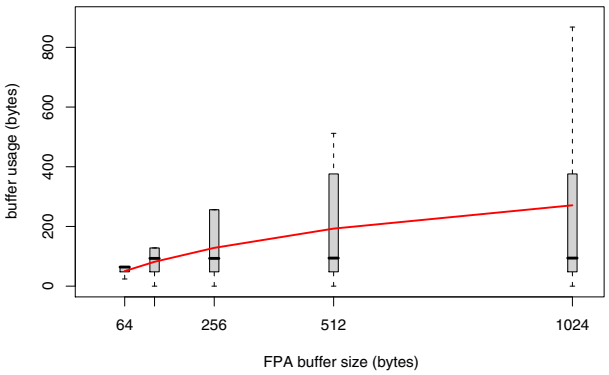


Fig. 9. FPA buffer usage in live dump with different FPA buffer sizes

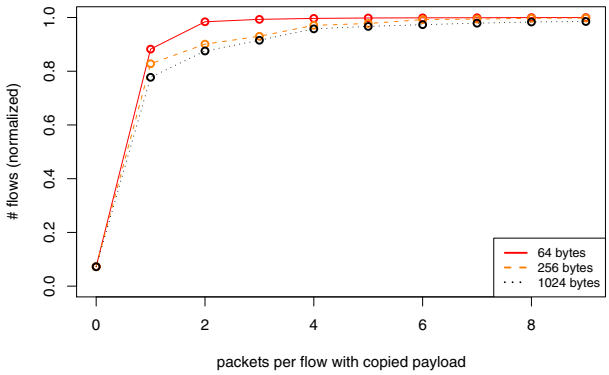


Fig. 10. Distribution of the number of packets where payload was accessed

the produced flow records and their corresponding templates. Thus, FPA buffers inside the flow records always have a static size. Depending on the length of the monitored flow, this buffer may be filled either completely or only partially. Our first analysis examines the actual number of bytes used in the flow records' FPA buffers. We configured Vermont to use 1024 Byte FPA buffers and fed all 5 live dumps into the flow meter. Figure 8 shows the results. We marked the median in this CDF with the dashed horizontal line. We notice a high variance in this median, with L_{IRC} featuring the lowest with 51 Byte and L_{HTTP} with 701 Byte. IRC traffic is usually completely interactive and so transfers very little data, whereas HTTP traffic does not seem to produce flows with little payload (see also Figure 6). L_{SSH} shows a large bump at 500 Byte, which is also visible in the connection size distribution at 1.7 kByte. The connection size distribution graph also includes header size and packets not carrying any application layer payload – this results in the difference of 1.2 kByte between the Figures. L_{ALL} has a low median of 94 Byte, as this trace also includes UDP packets and DNS name resolution packets, which have very low flow sizes.

To directly investigate the FPA buffer usage in relation to the buffer size f_{maxlen}, we replayed the live dump L_{ALL} to Vermont multiple times with varying buffer sizes and recorded the FPA payload usage. The results are shown in Figure 9. The maximum is always f_{maxlen} and the median stays at 94 Byte for the larger buffer sizes, like was already shown in Figure 8. The mean payload usage falls from 78 % relative to the maximum buffer size at f_{maxlen} =64 Byte down to 26 % at f_{maxlen} =1024 bytes.

3) Performance: In order to achieve high processing performance, the buffer size that enables FPA to ignore most packet's payload is limited. We analyzed the number of packets whose payload was accessed per flow in our complete live dump. Figure 10 presents the results for different FPA buffer sizes. Less than 10 % of all flows did not transfer any payload, so 0 packets are accessed. For the majority of all flows, only 1

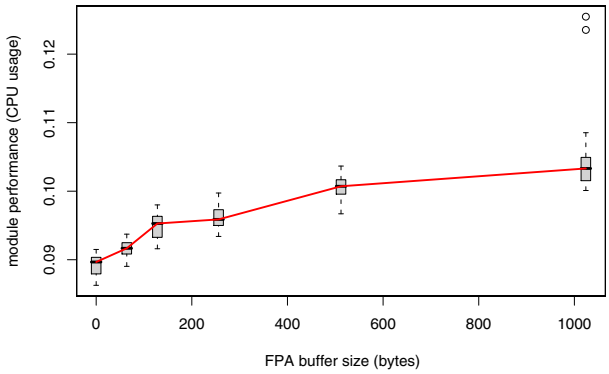


Fig. 11. CPU usage of PacketQueue for varying FPA buffer sizes

packet needs to be accessed. A buffer size of 1024 Byte shows only a slightly higher average amount of packets with accessed payload compared to a buffer size of 64 Byte. A flow contains in average 17.5 packets and the average number of packets with accessed payload for FPA buffer sizes of 64, 256, and 1024 Byte is 6.1 %, 7.9 %, and 9.4 % relative to each flow's total packet number, respectively.

Furthermore, we performed direct analysis of the effects of FPA to Vermont's live performance. Figure 11 shows the average CPU utilization of the PacketQueue module. This module performs aggregation of incoming packets to flow records and inserts the elements into the flow hash table. We varied the FPA buffer size between 0 (no FPA) and 1024 Byte. FPA directly influences the performance of this module: almost a linear increase in utilization can be seen when the FPA buffer size is increased to 128 Byte. After that, the effects of the falling relative FPA buffer usage can be seen and the module's performance increases in smaller steps. In total, 1024 Byte FPA increased the aggregation costs by only 15 %, which proves to be only a minor performance impairment. Figure 12 shows the

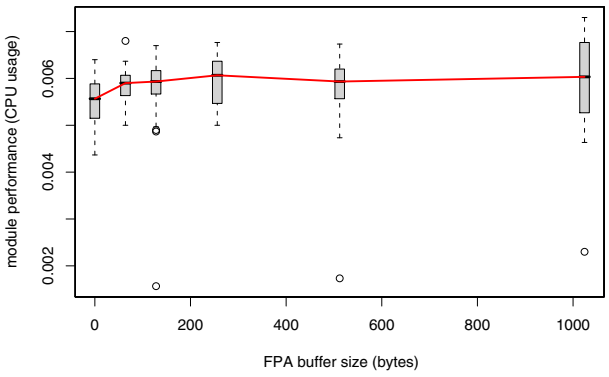


Fig. 12. CPU usage of PacketAggregator for varying FPA buffer sizes

CPU utilization of the PacketAggregator module. This module performs the removal of entries inside the hashtable and exports the flow records to the network. No dependency on the FPA buffer size is visible. This is mainly due to the fact that no copies of the contents of the flow records need to be performed any more and thus its performance stays at the same level.

V. Conclusion

We presented an extension to current flow monitoring techniques that we named Front Payload Aggregation (FPA). It allows the inclusion of the payload information within flow data. In particular, the first N Bytes of all the packets belonging to the flow are captured and added to the flow; exploiting transport layer information, i.e. the TCP sequence number, reordered packets can be corrected in the aggregator. This enables attack detection systems to separate intrusion detection from network sensors, i.e. the monitoring probes. The main goals during the design of FPA have been efficient aggregation and easy integration into current monitoring protocols. Our implementation is directly integrated into the software-based flow meter Vermont, which has been one of the first monitoring toolkits supporting the IPFIX protocol. During our extensive performance tests using real-live and artificial traffic dumps, we determined that FPA causes only an insignificant performance impact as payload for 256 Byte buffers is only extracted from in average 1–3 packets per flow.

According to our experiments, FPA only leads to a marginal performance overhead (CPU, memory). As we explore the performance of the PC hardware and operating system at its limits, it is not possible to exactly quantize this overhead due to side-effects of context switches and interrupt handling. However, the shown results clearly indicate that the implemented system performs well on commodity hardware.

Currently we are working on a modified version of Snort to evaluate at which position signatures match within a flow.

References

[1] 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.

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

[3] J. Jung, V. Paxson, A. W. Berger, and H. B. lakrishnan, "Fast Portscan Detection Using Sequential Hypothesis Testing," in *IEEE Symposium on Security and Privacy*, Berkeley/Oakland, CA, May 2004.

[4] C. Estan, S. Savage, and G. Varghese, "Automatically Inferring Patterns of Resource Consumption in Network Traffic," in *ACM SIGCOMM 2003*. Karlsruhe, Germany: ACM, August 2003, pp. 137–148.

[5] G. Carle, F. Dressler, R. A. Kemmerer, H. Koenig, C. Kruegel, and P. Laskov, "Network attack detection and defense - Manifesto of the Dagstuhl Perspective Workshop," *Springer Computer Science - Research and Development (CSRD)*, vol. 23, no. 1, pp. 15–25, March 2009.

[6] G. Schaffrath and B. Stiller, "Conceptual Integration of Flow-Based and Packet-Based Network Intrusion Detection," in *Resilient Networks and Services, Second International Conference on Autonomous Infrastructure, Management and Security (AIMS 2008)*, ser. Lecture Notes in Computer Science, vol. 5127. Bremen, Germany: Springer, July 2008, pp. 190–194.

[7] G. Muenz, N. Weber, and G. Carle, "Signature Detection in Sampled Packets," in *Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2007)*, Toulouse, France, November 2007.

[8] W. Willinger, M. S. Taqqu, R. Sherman, and d. V. Wilson, "Self-similarity through high-variability: statistical analysis of ethernet LAN traffic at the source level," in *the conference on Applications, technologies, architectures, and protocols for computer communication*. Cambridge, MA: ACM, 1995, pp. 100–113.

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

[10] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy, "Flow processing and the rise of commodity network hardware," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 39, no. 2, pp. 20–26, April 2009.

[11] 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.

[12] ——, "Predicting the resource consumption of network intrusion detection systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 437–438, 2008.

[13] R. Sommer and V. Paxson, "Exploiting Independent State For Network Intrusion Detection," in *21st Annual Computer Security Applications Conference (ACSAC 2005)*, 2005, pp. 59–71.

[14] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," in *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*, ser. Lecture Notes in Computer Science, vol. 5230. Cambridge, MA: Springer, September 2008, pp. 116–134.

[15] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Payson, and F. Schneider, "Enriching Network Security Analysis with Time Travel," in *ACM SIGCOMM 2008*. New York, NY, USA: ACM Press, August 2008, pp. 183–194.

[16] U. Shankar and V. Paxson, "Active Mapping: Resisting NIDS Evasion without Altering Traffic," in *IEEE Symposium on Security and Privacy*. Oakland, CA: IEEE, 2003, pp. 44–61.

[17] 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.

[18] S. Dharmapurikar and V. Paxson, "Robust TCP stream reassembly in the presence of adversaries," in *14th USENIX Security Symposium*, Baltimore, MD, July 2005.

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

[20] B. Claise, A. Johnson, and J. Quittek, "Packet Sampling (PSAMP) Protocol Specifications," IETF, RFC 5476, March 2009.

[21] J. Postel, "Transmission Control Protocol," RFC 793, September 1981.