Packet Sampling for Worm and Botnet Detection in TCP Connections

Lothar Braun, Gerhard Münz, Georg Carle
Institute for Informatics – Chair for Network Architectures and Services
Technische Universität München, Germany
Email: {braun|muenz|carle}@net.in.tum.de

***Abstract*—Malware and botnets pose a steady and growing threat to network security. Therefore, packet analysis systems examine network traffic to detect active botnets and spreading worms. However, with the advent of multi-gigabit link speeds, capturing and analysing header and payload of every packet requires enormous amounts of computational resources and is therefore not feasible in many situations.**

We address this problem by presenting an efficient packet sampling algorithm that picks a small number of packets from the beginning of every TCP connection. Bloom filters are used to store the required connection state information with constant amount of memory. Our analysis of worm and botnet traffic shows that the large majority of attack signatures is actually found in these packets. Thus, our sampling algorithm can be deployed in front of a detection system to reduce the amount of inspected packets without degrading the detection results significantly.

I. Introduction

Signature-based network intrusion detection systems (NIDS) have been in use for a long time. A well-known example is *Snort* [1], which acts as a network sniffer and analyses network traffic in order to detect attacks and other types of undesired traffic. Therefore, packet headers as well as packet payload are checked with pattern matching techniques against a database of attack signatures.

Signature detection becomes more and more complex because the number of attacks to search for increases, and because the signatures themselves become more sophisticated. For example, regular expressions are quite common in today's *Snort* rules. On the other hand, the amount of packets to be inspected increases with the permanent growth of network traffic. Hence, the resources of a single system are often not sufficient to analyse the entire traffic on a high speed link. As a result, random packet losses are likely to occur if the traffic exceeds the capacity of the detection system.

To keep track with large amounts of traffic, it is possible to increase the processing capacity, for example by deploying specialised signature matching hardware [2] or by distributing packets to multiple systems [3]. The disadvantage of these solutions is that they are very costly. As an alternative, we concentrate the available computing power on analysing the most relevant part of network traffic. For this purpose, we present a new sampling algorithm which selects packets carrying the first N payload bytes of every TCP connection. We expect these packets to contain sufficient information for detecting worm and botnet traffic. This approach is motivated by previous work which showed that the first packets of a connection or flow are the most relevant for detecting attacks [4] as well as for security forensic [5]. On the other hand, clipping long TCP connections by discarding the later packets dramatically reduces the amount of traffic to be analysed.

We analysed a large number of worm and botnet packet traces. The results confirm that the majority of today's worm and botnet traffic can be effectively found by inspecting the early packets of each connection. Of course, this might change in the future, so malicious content could be transmitted after a long series of legitimate data in order to evade detection. We will discuss this problem in Section V-C

The accurate identification of packets at the beginning of a TCP connection is not trivial if only passive traffic measurement data is available. It requires mechanisms for TCP connection tracking and TCP stream reassembly which are computationally complex and require a lot of memory to save connection states and to buffer out-of-order packets. Storing the connection states in a hash table may lead to memory exhaustion and loss of connections if the number of simultaneous TCP connections is very high, for example during TCP scans. Furthermore, possible collisions in the hash functions increases the complexity of inserting, querying, and deleting hash table entries. Hardware-based solutions have been presented to cope with these problems [6], yet the implementation of such solutions is very expensive.

We developed and implemented a novel sampling algorithm for deployment in high-speed networks with very high packet rates and large numbers of simultaneous TCP connections. The algorithm selects packets containing the first N payload bytes of a TCP connection by using a simplified TCP connection tracking mechanism and Bloom filters to store the connection states. Compared to existing solutions, the processing complexity per packet is reduced and the required amount of memory is constant at runtime. As a downside, sampling errors may occur because of the simplifications in the connection tracking mechanism as well as due to collisions in the hash functions of the Bloom filters. However, our evaluation of the sampling algorithm shows that high sampling accuracy can be achieved if the size of the Bloom filters is dimensioned appropriately.

In Section II, we introduce existing work related to packet sampling for the purpose of payload analysis. Additionally, this section gives an overview on the application of Bloom

filters in the area of networking. In Section III, we apply signature detection to real traces of worm and botnet traffic and examine how a restriction to the early packets of each TCP connection impacts the detection results. This examination confirms the initial assumption that the large majority of signatures is found in the first bytes of payload of a TCP connection. Section IV presents our packet sampling algorithm, including a description of the deployed simplified TCP connection tracking mechanism and Bloom filter variants. In Section V, we evaluate the probability of sampling errors by applying the sampling algorithm to real traffic traces. Finally, Section VI summarizes the advantages and limitations of the proposed algorithm and gives an outlook on possible extensions.

II. Related work

Kornexel et al. introduced the *Time Machine* [5] which allows storing network traffic from high-speed networks for a longer time period. In order to record large numbers of flows with limited storage capacity, only a few kilobytes from the beginning of each (unidirectional) flow are saved. This significantly reduces the amount of disk space due to the heavy-tailed flow length distribution. The authors state that most of the relevant information necessary for security forensic is preserved in the retained data, yet no evidence is provided that this assumption is valid. Maier et al. propose to combine the *Time Machine* with an intrusion detection system in order to correlate ongoing traffic with past observations [7].

The importance of the first packets of a flow has been shown in the context of attack detection and traffic classification. Wang et al. present PAYL, a system that searches for anomalies in the payload of network packets in order to detect attack traffic [4]. An evaluation based on real traffic traces shows that the detection rate does not decrease significantly if only the first 1000 bytes of payload of each flow are analysed instead of complete flows. On the other hand, processing time drops dramatically due to the reduced amount of analysed traffic. Regarding traffic classification, Sen et al. are able to classify a flow as belonging to a P2P application after looking at the first ten packets [8]. Won et al. show that most application signatures appear within the first five packets of a flow [9]. The authors ascertain that analysing all packets of the flow may even lead to misclassifications which do not occur if only the first packets are examined.

The utilization of Bloom filters [10] and similar data structures has been proposed for various traffic measurement purposes. Chang et al. use Bloom filters to store recent packet classification results [11]. Kong et al. use Time-out Bloom filters (i.e., Bloom filters saving timestamps) to sample the first packet of every observed flow [12]. The same authors combine the Time-out Bloom filter with a Count-Min Sketch (CMS), which is a Bloom filter storing counters, to detect port scans [13]. The usage of CMS has also been proposed for counting the number of distinct flows [14], [15], for counting the number of packets or bytes per flow [16]–[18], and for distributed traffic monitoring [19].

Whitehead et al. memorize every observed TCP SYN packet in Bloom filters to detect established TCP connections [20]. Only packets belonging to an established TCP connection are sampled, yet without limiting the number of sampled packets. Close to our approach is the work of Canini et al. where a chain of Bloom filters is used to sample the first J packets of every bidirectional flow [21]. The sampling limit is expressed as the maximum number of packets, which works fine for classifying most kinds of legitimate traffic but makes it easy for an attacker to circumvent packet selection, for example by sending empty TCP ACK packets.

Our packet sampling algorithm aims at sampling the first N payload bytes of each TCP connection and thus approximates the behaviour of the *Time Machine*. In contrast to the *Time Machine* and Canini's approach, our algorithm tracks the establishment and termination of every TCP connection, which enables us to remove the saved information of terminated connections. Hence, we do not need to reset the Bloom filters periodically. Moreover, we demonstrate the usability of our sampling strategy for signature-based worm and botnet detection by examining real worm and botnet traffic.

III. Worm and Botnet Detection in First Packets

The basic reasoning behind focusing on the packets at the beginning of a flow is that these packets usually contain sufficient information to classify the entire flow as harmful or benign. In this section, we evaluate to which extend this assumption is true for worm and botnet traffic. Therefore, we use *Snort* [1] to analyse traffic of different worms and botnets. We determine the positions within the payload of the TCP session at which the matching signatures are found. As a result, we can evaluate how the detection results are affected if the analysis is restricted to a few kilobytes of payload.

In contrast to the *Time Machine* [5], which restricts the number of sampled bytes on the basis of unidirectional flows, our sampling algorithm jointly counts the number of TCP payload bytes exchanged in both direction of a TCP connection. For TCP connections, this approach is appropriate for NIDS performing session analysis in addition to the inspection of individual packets. As the traffic volume in both direction is usually not symmetric, separate limits for both directions would result in an earlier cut-off of one direction. After the cut-off is reached in one direction, an IDS cannot properly continue session analysis as one direction of traffic is missing.

Our analysis is based on traffic of 93 different worms and bots, which have been run in a controlled environment. Every worm sample has been executed several times at different days in order to see different behaviours based on the commands received from the botmaster. The detection is done with two rule sets: the standard rule set shipped with *Snort* 2.8.4.1 and the ruleset downloaded from emergingthreats.net [22] on June 11th 2009. These rules detect a multitude of events ranging from Command&Control (C&C) channels of different bots, shell code, exploits, bot downloads, etc. Altogether, 4030 alarms are raised, among which 3511 alarms are triggered by TCP packets, 451 by UDP packets, and 68 by ICMP packets.

TABLE I
OVERVIEW OF TCP ALARMS

Alarm	Frequency
Shipped rule set:	
- Shellcode	45
- Backdoor/Spyware	38
- HTTP oversize chunk encoding	12
Emergingthreats rule set:	
- Instant Messaging	1116
- HTTP	810
- Mail	94
- Scanning	1041
- Miscellaneous	355

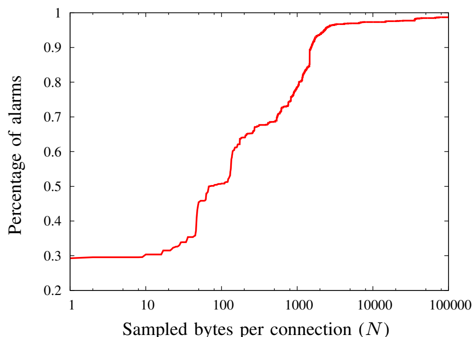

Fig. 1. TCP alarms depending on number of payload bytes

For our evaluation, we are interested in the TCP alarms only which constitute the majority of alarms.

Only 95 alarms are raised by the shipped rule set while 3416 alerts go back to the emergingthreats rule set. Table I shows the alarm frequency for different causes. Alarms described as 'HTTP oversize chunk encoding' are triggered if the chunk length indicated in an HTTP header is larger than the data provided. Most of the emergingthreats alarms are related to scanning activity and C&C channels. C&C channels are realised on top of Instant Messaging protocols, such as IRC, HTTP, or SMTP. Further alarms found in HTTP traffic originate from malware downloads. Finally, a large number of TCP scans are detected, which is a common method for a worm to find new victims. The observed scans are directed to the ports of specific applications, such as VNC (5000-5020), NetBIOS and SMB (135, 139, 445). 'Miscellaneous' contains all remaining alarms which do not fit in any of the other categories.

We calculate the position within the TCP connection at which an alarm is raised using the TCP sequence and acknowledgement numbers as well as TCP payload length of the triggering packet. Hence, the calculated values are not byte accurate but aligned with packet boundaries. This corresponds to the result of our algorithm which samples traffic on a per-packet basis.

Figure 1 displays the percentage of TCP alarms as a function of the number of payload bytes N analysed. As can be seen, most of the alarms are raised very early. About 30% of all alarms are triggered by scans and therefore only need the handshake packets for detection. Most of the alerts (96%) are found within less than 3kB of payload from the beginning of the connections. This means that a sampling limit of a few kilobytes should be sufficient to detect the large majority of today's worm and botnet traffic. The last alarms are found at about 682kB after the beginning of the connection. All of them belong to a generic shellcode rule.

IV. CONNECTION BASED PACKET SAMPLING

This section describes the proposed sampling algorithm which selects those packets carrying the first N bytes of payload of every TCP connection, which requires an appropriate TCP connection tracking mechanism. The next subsection sketches the general challenges of TCP connection tracking and discusses the complexity of accurate TCP stream reassembly. In Section IV-B, we present a simplified connection tracking scheme which is more appropriate for connection-based packet sampling in high-speed networks as it maintains much less state information per connection. Section IV-C introduces two Bloom filter variants used in our packet sampling algorithm. The implementation details are given in Section IV-D, where we will describe how to deploy these Bloom filters and how to use them to store TCP connection states. Finally, we summarize the main properties of the proposed sampling algorithm in Section IV-E

A. Accurate TCP Connection Tracking

TCP connections are identified by the addresses and ports of both communication endpoints and the protocol identifier. Start and end of a regular TCP connection are defined by SYN, FIN, and RST packets that are exchanged during the TCP connection establishment and the termination phase. User data is transferred during the communication phase. With help of sequence numbers, the receiving peer is able to detect reordering, loss, and duplication of packets in the network. Packet losses are usually rectified by retransmissions.

TCP connection tracking aims at deducing the connection state as well as the exchanged data by analysing TCP traffic which has been passively monitored in the network. A complete analysis of a TCP connection requires to detect the connection establishment and termination, to recognize duplicate packets, and to reorder the packets according to the sequence numbers. Problems may occur if the monitoring process does not observe all packets of the TCP connection. This may happen due to routing changes or asymmetric routing. Additionally, it occurs quite frequently that TCP connections are not shut down properly because of connectivity problems or because one of the communication endpoints disappears without terminating the connection (e.g., due to a system crash). In such situations, an external observer does not know if and when the connection endpoints consider the TCP connection as timed out.

Apart from these general problems, performing accurate TCP connection tracking is quite complex and consumes a certain amount of processing and memory resources for each connection. The accurate reassembly of the exchanged data

requires to maintain more or less the same information that is also kept by one of the connection endpoints. In particular, sequence numbers need to be examined to reorder out-of-order packets and to detect duplicates. Furthermore, a buffer is needed to withhold a packet until all preceding packets have arrived. Consequently, accurate connection tracking does not scale well if the number of simultaneous TCP connections is very large, as it is the case in high-speed networks. Also, we have to cope with exceptional situations resulting in an increased number of TCP connections. Examples are TCP port and network scans as well as SYN flooding attacks, causing large numbers of so-called half-open connections which have to be saved by the tracking mechanism until a time-out can be assumed.

If the exact reassembly of TCP connections is not necessary, it is possible to simplify TCP connection tracking in order to improve scalability. We present such a solution in the next subsection.

B. Simplified TCP Connection Tracking

We improve the scalability of TCP connection tracking by reducing the amount of state information that has to be kept per connection as well as the processing complexity per packet. The goal is to keep the decision whether to select or discard a packet as simple as possible while maintaining the sampling accuracy at an acceptable level. Hence, the proposed solution trades sampling accuracy off against scalability and high packet throughput.

The first simplification concerns the detection of connection establishments. Instead of looking for complete TCP three-way handshakes, we consider one SYN packet shortly followed by a second packet without SYN flag as indicator of a successfully established TCP connection. Both packets have to be exchanged between the same endpoints identified by tuples of IP address and port number. We ignore the direction of these two packets, which allows us to detect the beginning of a connection even if we observe only one direction of the traffic, for example due to asymmetric routing. The two packets have to be observed within a small time interval. We determined three seconds to be an appropriate value for this parameter from network traces. Hence, if only a SYN packet is observed, the corresponding state will be automatically deleted after this time-out. There is a small risk of handshake detection errors because we do not check if the sequence and acknowledgement numbers of the two packets are plausible.

The second simplification concerns the connection reassembly: we do not perform any packet reordering nor do we remove duplicated packets. We leave these tasks to the subsequent packet analysis step (e.g., an NIDS) for which knowledge about wrong packet orders and duplicates may even be of interest. Hence, omitting these task in the packet sampling algorithm does not necessarily represent a disadvantage. More important is the influence on the sampling results. In order to sample those packets containing the first N bytes of payload, we count the payload lengths in the order of packet arrival without regarding sequence numbers. In the presence of packet reordering and duplicates, we risk selecting packets which should not be sampled (false positives) and risk dropping packets which should be sampled (false negatives). However, we expect that these problems are not very frequent at the beginning of a TCP connection.

Finally, we consider a TCP connection to be terminated if we observe a FIN or RST packet. Again, we do not check if the packet's sequence number is valid, which may cause a problem. We drop all packets observed after the FIN or RST packet. This may result in false negatives if the sampling limit is not reached and if more data is sent from the other peer before shutting down the connection. In order to clean up stale TCP connections which have not been terminated properly, an supplementary time-out mechanism needs to be implemented which removes the state of connections with long idle times.

As we have seen, the proposed simplifications may lead to wrong sampling decisions under certain conditions. In Section V-B, we evaluate the frequency of false positives and false negatives by comparing the results of accurate and simplified connection tracking applied to real traffic traces.

The simplified connection tracking mechanism still needs to maintain certain state information for every connection. We need to store the arrival of the first SYN packet as well as the counter containing the number of payload bytes to be sampled. The counter must be kept as long as packets are sampled from the corresponding connection, which means until one of the following happens: a FIN or RST packet is observed, the sampled packets reach the configured number of payload bytes, or the connection is timed out.

Although the amount of state data is much smaller than in the case of accurate TCP connection tracking, the required memory still linearly depends on the number of simultaneous TCP connections. As mentioned earlier, the number of TCP connections may grow to very large numbers in high-speed networks or in certain attack situations. To solve this problem, we make use of Bloom filters in which we store the connection states. As a result, the algorithm operates with a fixed amount of memory which is independent of the number of parallel connections. The next section introduces the utilized types of Bloom filters before Section IV-D describes their deployment.

C. Bloom filters

A Bloom filter [10] is a probabilistic data structure that is capable to store information about a set of elements without storing the elements itself. In particular, the stored information specifies whether an element is part of the set or not. The filter is composed of a bit array and a set of hash functions which index the individual bits. Initially, every bit in the array is set to zero, indicating that the set is empty. If a new element is to be inserted into the set, all hash functions must be calculated for this element in order to get all associated indexes. All corresponding bits are then set to one and the new element is considered as part of the set. If at least one of the bits is zero, the element is not part of the set.

False positives are possible due to collisions in the hash functions, meaning that an element can be identified as part of

the set even though it has not been added to it. This happens if, and only if, all bits associated to the queried element have been set by other elements. False negatives (i.e., elements which have been added to the set but cannot be identified by the Bloom filter) are not possible.

Memory consumption can be calculated depending on the number of used hash functions (l), the collision probability of the hash functions (p) and the number of stored elements (k) [11]:

$$m = -\frac{l \cdot k}{\ln\left(1 - p^{\frac{1}{l}}\right)} \quad (1)$$

A drawback of conventional Bloom filters is that elements cannot be deleted from the set. It is only possible to reset the entire filter, which results in a complete loss of the stored information. For connection tracking, however, we must be able to remove the connection state after the connection terminated. Therefore, we use two Bloom filters variants summarized in the following.

The first variant is called Time-out Bloom filter [12]. Its array is composed of timestamps instead of bits. In order to insert an element into the filter, the associated timestamps are overwritten with the current time. Each element is only valid for a certain amount of time and will automatically expire after this time span. To check if an element is still valid, the difference between the current time and the oldest associated timestamp in the filter is compared to the given timeout.

The second Bloom filter variant is called Count-Min Sketch (CMS) [23] and associates a positive value to each inserted element. Its array consists of counters which can be increased and decreased. With the insertion of an element, a positive value is added to the associated counters. When querying an element, the smallest associated counter contains the current value of the element. If the smallest counter is zero, the element is not considered as part of the set.

We use these two filter variants to store the connection states needed to perform simplified TCP connection tracking and to sample the first N bytes of payload per TCP connection.

D. The Algorithm

We implemented a packet sampling algorithm which selects the first packets of a TCP connection until a maximum of N bytes of payload has been exported. The algorithm uses the simplified TCP connection tracking mechanism presented in Section IV-B and uses two Time-out Bloom filters and one CMS to store the required connection states. We use 2-universal hash functions [24] for the Bloom filters.

Each TCP connection is identified by the quadruple of source IP address (SA), destination IP address (DA), source port (SP), and destination port (DP). The element stored in the filters results from the following concatenation:

$$\min\{(SA||SP),(DA||DP)\}\,||\,\max\{(SA||SP),(DA||DP)\}$$

As we sort the tuple of IP address and corresponding port numerically, both directions of the TCP connection map to the same element.

The first Time-out Bloom filter stores the timestamps of all observed SYN packets. The CMS stores the number of payload bytes which need to be exported for an established TCP connection. The second Time-out Bloom filter stores the point in time at which the packet sampling for a given TCP connection was stopped, either because the maximum number of payload bytes was reached or because a FIN or RST packet was observed. The three filters are called *start filter*, *export filter*, and *stop filter* in the remainder of this paper.

The packet treatment of the sampling algorithm can be summarized as follows:

On the arrival of a SYN packet, the timestamp of the packet is written into the start filter and the packet is sampled.

For any packet which is not a SYN, FIN, or RST packet, we first check, if a corresponding connection exists in the export filter (non-zero value). If this is the case, the packet is sampled and the counters in the export filter are decreased by the minimum of the payload length and the element's value currently stored in the filter. This is to ensure that the stored value does not become negative.

If the connection does not exist in the export filter, we look up the timestamps stored in the start and stop filters. If the timestamp of the start filter is more recent than the timestamp in the stop filter, and if the timestamp in the start filter is not older than three seconds, the packet belongs to a new connection for which packet sampling should be started. Hence, the packet is sampled and the maximum number of payload bytes N to be sampled minus the payload length of the packet is inserted into the export filter. In any other case, the packet is not sampled.

On the arrival of a FIN or RST packet, we check if the connection exists in the export filter. If this is the case, the packet is sampled and the connection is deleted from the export filter by subtracting the current element's value from the associated counters. In addition, the timestamp of the packet is saved in the stop filter. If the connection is not included in the export filter, the FIN or RST packet is not sampled.

As already mentioned, collisions may lead to corrupt information stored in the filters. Consequences are sampling errors in the form of packets which are sampled although they should not (false positives) and packets which are dropped although they should be sampled (false negatives). In Section V-B, we assess the number of sampling errors by applying the algorithm to real traffic traces. Furthermore, we evaluate how many errors are caused by the simplified TCP connection tracking mechanism and how this number increases due to collisions in the hash functions of the Bloom filters.

E. Main Properties of the Sampling Algorithm

Our sampling algorithm approximately samples the first N bytes of payload per TCP connection. Therefore, it implements a simplified TCP connection tracking mechanism which requires less memory per connection state than accurate connection tracking. Since the connection states are stored in Bloom filter structures, the memory consumption stays constant during runtime. The computational complexity per packet

TABLE II
PROPERTIES OF NETWORK TRACES

Property	Twente	Munich
Duration (minutes)	15	87
Packets	16,714,065	10,810,511
TCP packets	11,534,706	10,491,400
TCP connections	35,413	18,524
Size	10.2 GB	10.9 GB
TCP data	8.9 GB	10.7 GB

is constant and does not depend on the number of simultaneous TCP connections or the filling level of the filters. Stale connections may lead to false positives in other connections, yet do not occupy any additional memory. Similarly, TCP scans and SYN flooding attacks fill the start filter and may result in false positives in other connections. However, the required memory remains constant again. In general, the collision probability increases with increasing number of simultaneous connections stored in the filters, which may degrade the sampling accuracy depending on the order of packet arrival.

The *Time Machine* [5] stores a few kilobytes per unidirectional flow, counting packet headers and payload. In contrast, we restrict the total number of payload bytes sampled in both direction. Hence, if the sampling limit is reached, sampling stops for the entire TCP connection, which is useful if both directions are jointly analysed, such as in an NIDS. Furthermore, counting payload lengths instead of packet lengths is advantageous since empty packets will be selected without decreasing the number of bytes still to be sampled. As the *Time Machine* uses hash tables to store the state of each flow, it requires variable amount of memory and does not scale well if the number of simultaneous flows becomes very high.

The main difference to Canini's approach [21] is that we count payload bytes instead of packets. Since byte counters are several magnitudes larger than packet counters, Canini's approach cannot be adopted to achieve our sampling goal.

V. EVALUATION

In this section, we evaluate the accuracy of the proposed sampling algorithm by applying it to real traffic traces. Section V-A gives some details about the traces. In Section V-B, we determine which kind of errors and how many of them actually occur depending on different parameter settings. Finally, we discuss possible ways to evade packet sampling in Section V-C.

A. Traces Used in Evaluation

We evaluate our algorithms using two traffic traces. The first one was captured at the access link of a student residence at the University of Twente [25]. The second trace file has been recorded in the network of our research group at the Technische Universität München. The two traces will be called *Twente trace* and *Munich trace* in the following. Some properties of the traces are listed in Table II.

We implemented a reference algorithm which provides accurate TCP connection tracking, packet reordering and removal of duplicate packets to identify those packets which

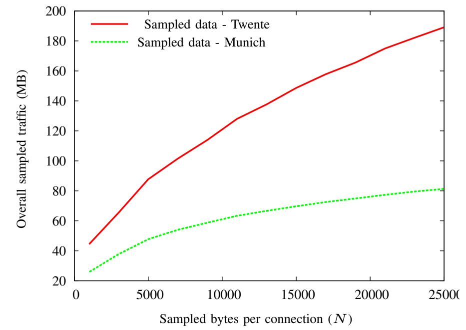

Fig. 2. Amount of sampled traffic

should be sampled depending on the sampling limit N. Figure 2 shows the amount of traffic (including packet headers) sampled by the reference algorithm. As expected, the overall amount of sampled traffic increases for larger values of N. However, the amount of sampled traffic is very small compared to the amount of TCP traffic in the original traces, which confirms the observations of previous work [5]. In the case of the Twente trace, between 476,817 (N = 1kB) and 512,334 (N = 25kB) packets are sampled. The amount of sampled TCP data is between 44 MB and 189MB, representing 0.5% to 2.1% of the TCP traffic volume. In the case of the Munich trace, the data reduction is even larger: Between 314,063 and 326,742 TCP packets are sampled, corresponding to only 25MB (about 0.2%) to 81MB (about 0.8%) of the TCP traffic. This was expected because the Munich trace contains about half as many TCP connections as the Twente trace at a comparable number of TCP packets. Thus, TCP connections in the Munich trace are longer and contain more packets on average, which increases the effect of the packet sampling.

B. Empirical Analysis of Sampling Errors

In Section IV-B, we discussed under which conditions the simplified connection tracking mechanism may cause sampling errors. Additional errors may be caused by the utilization of Bloom filters due to collisions in the hash functions. The goal of our evaluation is to assess how many sampling errors are caused by simplified connection tracking and how many errors go back to the Bloom filters. Therefore, we implemented a second variant of the algorithm described in Section IV-D which stores the same connection state information in hash tables with concatenated lists instead of Bloom filters. In various experiments, we determined the numbers of false positives and false negatives for both implementations, taking the output of the reference algorithm as ground truth.

Since the number of hash functions influences the computing cost, a small number of hash functions is desired. On the other hand, a large number of hash functions reduces the probability of collisions. We found that $l = 3$ hash functions is a good trade-off between computational complexity and collision probability.

We dimension the Bloom filters for the expected number of connections to be stored simultaneously. In practice, this

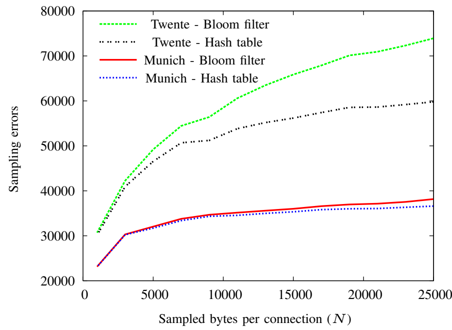

Fig. 3. Influence of sampling limit N

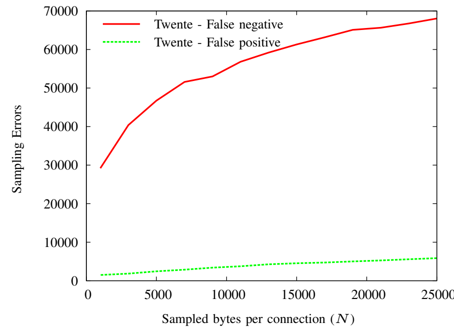

Fig. 4. False positives vs. false negatives

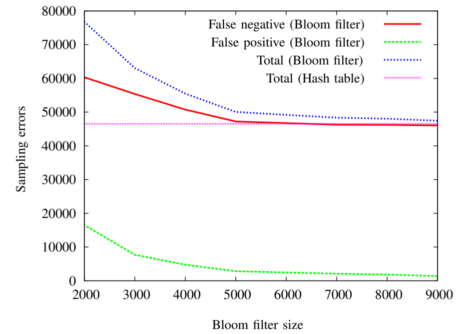

Fig. 5. Influence of Bloom filter size

value needs to be estimated or determined empirically. For our evaluation, we can obtain it with help of the hash table based variant of our sampling algorithm. The maximum number of simultaneous connections that need to be stored in the hash table is 4573 for Twente trace and 1150 for Munich trace. On average, however, only about 1168 and 248 connections are stored, respectively.

Using equation (1), we calculate the necessary minimum filter sizes for which the expected collision probability does not exceed 10% ($p = 0.1$). Based on the results, we use filter sizes of 5800 entries for Twente trace and 1200 entries for Munich trace in the following.

We determine the overall number of sampling errors (false positives and false negatives) for both variants of the algorithm. Figure 3 shows the results in dependence of the sampling limit N. As can be seen, more sampling errors occur in the Twente trace than in the Munich trace, which can be explained by the larger number of TCP connections in the Twente trace. For both variants of the algorithm, the number of errors increases for larger N.

In the case of the hash table based algorithm, the errors may only result from simplified TCP connection tracking. With increasing N, more packets need to be selected by the sampling algorithm, thus more duplicate packets and wrongly ordered packets can turn into sampling errors. The error rate is small compared to the overall number of packets in the traces. If 1kB of payload is sampled per connections in the Twente trace, only 30,130 and 30,694 errors are produced by the hash table based variant and the Bloom filter variant of the sampling algorithm, respectively. This corresponds to 7.35% and 7.36% with respect to the number of sampled packets. With a sampling limit of $N = 25$kB, the numbers increase to 59,789 (11.6%) and 73,925 (14.4%). Regarding the Munich trace, the hash table based variant causes between 23,106 (6.3%) and 36,601 (11.2%) errors for $N = 1$kB and $N = 25$kB, respectively. With Bloom filters, the figures are 23,131 (6.4%) and 38,156 (11.6%). Apparently, the impact of collisions in the hash functions on the sampling results is smaller than for the Twente trace, which can be explained by a different traffic characteristics.

In summary, it can be said that most sampling errors go back to simplified TCP connection tracking. Such errors also occur with other sampling approaches not performing accurate TCP connection tracking, such as the *Time Machine*. The utilization of Bloom filters increases the sampling errors by 0.1% to 24%, depending on the traffic.

For the Twente trace, Figure 4 differentiates the sampling errors of the Bloom filter based variant into false negatives and false positives. Most of the errors are false negatives. The likely reason is that packets are retransmitted within the first N bytes of the connection.

Now, we evaluate how the number of errors evolves if smaller or larger Bloom filter sizes are chosen. Figure 5 shows the sampling errors for filter sizes in the range of 2000 to 9000 entries. The sampling limit is set to $N = 5$kB. As expected, smaller filter sizes result in more sampling errors, reaching 76,815 sampling errors for $m = 2000$. For larger filter sizes ($N > 9000$), the number of errors falls below 47,450 and approaches the number of errors of the hash based variant of the algorithm (46,548). This means that the impact of the Bloom filters becomes very small.

C. Evading Packet Selection

An attacker being aware of the sampling strategy could try to evade packet selection and detection in two different ways. Firstly, he could locate the malicious part of payload beyond the first N bytes of a TCP connection. This is a general problem which also affects similar approaches such as the *Time Machine*.

Secondly, he could trick the simplified TCP connection tracking mechanism by inserting packets with identical addresses and ports but invalid sequence numbers as described in Section IV-B. Although discarded by the receiver, such packets will be sampled if they are observed within the sampling limit, possibly causing the dropping of later packets with valid sequence numbers that should be sampled. Again, the *Time Machine* cannot cope with this kind of evasion, either.

To make evasion more difficult, we can dynamically vary N over time (e.g., depending on the current load on the detection system) to make the sampling limit less predictable. Furthermore, we can choose different values for N depending on some predefined filters, for example in order to sample fewer packets exchanged between trustworthy endpoints.

The attacker may also attack the sampling algorithm itself,

for example by performing a TCP scan in order to poison the values stored in the start filter. This may lead to connections being erroneously added to the export filter causing false positives and an increased sampling rate which risks to overload the analysis system. However, as all SYN packets are sampled, such an attack will be detected. Moreover, the disturbing effect on the sampling result is temporary, which means that the false positive rate will decline to normal level after the scan is over.

VI. CONCLUSION

We have presented a sampling algorithm which selects packets containing the first N payload bytes of each TCP connection. According to our analysis of real worm and botnet traffic with *Snort*, the large majority of the signatures are found within the first few kilobytes of payload. Hence, our algorithm can be deployed to reduce the load of the NIDS without degrading the detection results significantly.

The sampling algorithm makes use of a simplified TCP connection tracking mechanism and Bloom filters to store the required connection states. Hence, memory consumption remains constant during runtime, which means that packet losses due to memory exhaustion may not occur. Moreover, the computational complexity per packet is constant and independent of the observed traffic. Thus, the algorithm can be efficiently implemented in software or hardware to sample traffic at high speed links and pass the selected packets to a subsequent detection system.

An empirical evaluation of the sampling algorithm shows that the number of sampling errors is small and rarely affected by collisions in the Bloom filters if their sizes are appropriately dimensioned for the average number of simultaneous TCP connections. Under extreme conditions with an unexpectedly high number of connections, the number of sampling errors increases slowly without affecting the function of the sampling algorithm.

Although we have only considered TCP connections in this paper, it would be possible to develop a similar sampling algorithm for UDP traffic. In this case, however, we cannot profit from flags indicating the beginning and end of a connection.

ACKNOWLEDGMENTS

We gratefully acknowledge support from the German Research Foundation (DFG) funding the LUPUS project in the scope of which this research work as been conducted.

REFERENCES

[1] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," in *Proc. of 13th USENIX Systems Administration Conference (LISA'99)*, Seattle, WA, USA, Nov. 1999.

[2] S. Fide and S. Jenks, "A Survey of String Matching Approaches in Hardware," Dept. of Electrical Engineering and Computer Science, University of California, Irvine, Tech. Rep. TR SPDS 06-01, Mar. 2006.

[3] M. Colajanni and M. Marchetti, "A Parallel Architecture for Stateful Intrusion Detection in High Traffic Networks," in *Proc. of Workshop on Monitoring, Attack Detection and Mitigation (MonAM) 2006*, Tübingen, Germany, Sep. 2006.

[4] K. Wang and S. J. Stolfo, "Anomalous Payload-Based Network Intrusion Detection," in *Proc. of International Symposium on Recent Advances In Intrusion Detection (RAID) 2004*, Sophia Antipolis, France, Sep. 2004.

[5] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer, "Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic," in *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC) 2005*, Berkeley, CA, USA, Oct. 2005.

[6] S. Dharmapurikar and V. Paxson, "Robust TCP Stream Reassembly in the Presence of Adversaries," in *Proc. of the USENIX Security Symposium 2005*, Baltimore, MD, USA, Jul. 2005.

[7] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, "Enriching Network Security Analysis with Time Travel," in *Proc. of ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Seattle, WA, USA, Aug. 2008.

[8] S. Sen, O. Spatscheck, and D. Wang, "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures," in *Proc. of International Conference on World Wide Web*, New York, NY, USA, May 2004.

[9] Y. Won, B.-C. Park, H.-T. Ju, M.-S. Kim, and J. Hong, "A Hybrid Approach for Accurate Application Traffic Identification," in *Proc. of IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMon) 2006*, Vancouver, BC, Canada.

[10] B. H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, 1970.

[11] F. Chang, W. Feng, and K. Li, "Approximate Caches for Packet Classification," in *Proc. of Conference of the IEEE Computer and Communications Societies (INFOCOM) 2004*, Hong Kong, China, Mar. 2004.

[12] S. Kong, T. He, X. Shao, and X. Li, "Time-out Bloom Filter: A New Sampling Method for Recording More Flows," in *Proc. of International Conference on Information Networking (ICOIN) 2006*, Sendai, Japan, Jan. 2006.

[13] S. Kong, T. He, X. Shao, C. An, and X. Li, "A Double-Filter Structure Based Scheme for Scalable Port Scan Detection," in *Proc. of IEEE International Conference on Communications (ICC) 2006*, Istanbul, Turkey, Jun. 2006.

[14] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms for Counting Active Flows on High Speed Links," in *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC) 2003*, Miami Beach, Florida, USA, Oct. 2003.

[15] J. Sanjuas-Cuxart, P. Barlet-Ros, and J. Sole-Pareta, "Counting Flows over Sliding Windows in High Speed Networks," in *Proc. of IFIP Networking Conference*. Aachen, Germany: Springer, May 2009.

[16] A. Kumar, J. J. Xu, J. Wang, O. Spatschek, and L. E. Li, "Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement," in *Proc. of Conference of the IEEE Computer and Communications Societies (INFOCOM) 2004*, Hong Kong, China, Mar. 2004.

[17] A. Kumar and J. J. Xu, "Sketch Guided Sampling - Using On-Line Estimates of Flow Size for Adaptive Data Collection," in *Proc. of Conference of the IEEE Computer and Communications Societies (INFOCOM) 2006*, Barcelona, Spain, Apr. 2006.

[18] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani, "Fast Monitoring of Traffic Subpopulations," in *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC) 2008*, Vouliagmeni, Greece, Oct. 2008.

[19] G. Cormode, S. M. Muthukrishnan, and W. Zhuang, "Whats Different: Distributed, Continuous Monitoring of Duplicate-Resilient Aggregates on Data Streams," in *Proc. of IEEE International Conference on Data Engineering (ICDE) 2006*, Atlanta, Georgia, USA, Apr. 2006.

[20] B. Whitehead, C.-H. Lung, and P. Rabinovitch, "A TCP Connection Establishment Filter: Symmetric Connection Detection," in *Proc. of IEEE International Conference on Communications (ICC) 2007*, Glasgow, Scotland, Jun. 2007.

[21] M. Canini, D. Fay, D. J. Miller, A. W. Moore, and R. Bolla, "Per Flow Packet Sampling for High-Speed Network Monitoring," in *Proc. of International Conference on Communication Systems and Networks (COMSNETS) 2009*, Bangalore, India, Jan. 2009.

[22] Website of the project *Emerging Threats*, Internet: http://www.emergingthreats.net/.

[23] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and its Applications," *Journal of Algorithms*, vol. 55, no. 1, 2005.

[24] J. L. Carter and M. N. Wegman, "Universal Classes of Hash Functions," *Journal of Computer and System Science*, vol. 18, no. 2, 1979.

[25] Traffic repository Twente University, http://traces.simpleweb.org.