Counting Bloom Filters for Pattern Matching and Anti-Evasion at the Wire Speed

Gianni Antichi, Domenico Ficara, Stefano Giordano, Gregorio Procissi, and Fabio Vitucci, University of Pisa

Abstract

Standard pattern-matching methods used for deep packet inspection and network security can be evaded by means of TCP and IP fragmentation. To detect such attacks, intrusion detection systems must reassemble packets before applying matching algorithms, thus requiring a large amount of memory and time to respond to the threat. In the literature, only a few efforts proposed a method to detect evasion attacks at high speed without reassembly. The aim of this article is to introduce an efficient system for anti-evasion that can be implemented in real devices. It is based on counting Bloom filters and exploits their capabilities to quickly update the string set and deal with partial signatures. In this way, the detection of attacks and almost all of the traffic processing is performed in the fast data path, thus improving the scalability of intrusion detection systems.

Intrusion detection systems (IDSs) are devices that protect a network by analyzing all the ingoing traffic and detecting potentially malicious data. Standard detection methods consist of searching by using pattern matching algorithms in IP packets for predefined signatures that characterize an attack. The more recent techniques involve the use of finite automata (FA) or hybrid schemes such as Aho-Corasick-Boyer-Moore, but their performance is still limited by issues such as memory size and access delay.

Hence, for string matching, hardware architectures also were proposed. For instance, to reach very high speeds, FA in field programmable gate arrays (FPGAs) are used; however, for adding and deleting strings, a hardware reconfiguration is required, which is too expensive. The simplest approach is a ternary-content addressable memory (TCAM), which stores all the strings. Searches can be very fast (a single clock cycle); however, the high cost makes TCAM infeasible for large signature sets.

Recently, Bloom filters (BFs) also were used for pattern matching [1, 2]. They are hash-based structures that trade a certain degree of accuracy for considerable savings in memory. BFs were created to represent a set of elements and to perform membership queries so they can be adopted for pattern matching simply by constructing filters according to a set of signatures. The advantages are the compact representation that is typical of BFs and a remarkable reduction of the amount of traffic handled by the slow path, which result in a general performance improvement and scalability of IDSs. The work in [1] adopts parallel BFs: each of them represents the strings of a specific length, in this way enabling a fast search. The work in [2] instead combines BFs and parallel hashing: first packets are passed through a BF, which detects some strings by acting as an accelerator. Then, such strings are dispatched to the parallel hashing engine, which performs a hash comparison, and in the case of a hash hit, compares the input string to the actual string to eliminate any false positives.

Moreover, although the set of signatures to be detected changes very frequently (due to the continuous creation of new viruses and attacks), Bloom filters do not address the issues of changing items in a set; hence, counting Bloom filters (CBFs) were designed. They are based on the same principles as BFs but use counters to take into account the occurrences of items — in this way allowing for quickly updating the string set and identifying candidates to be used in pattern matching. The work in [3] proposes the use of a bit vector in which each bit corresponds to a counter of the CBF representing the string set. Whenever a member is added to or deleted from the CBF, the corresponding counters are increased or decreased, respectively. If a counter changes from zero to one, the corresponding bit in the bit vector is set, whereas it is cleared if the counter changes from one to zero. Because the counters are changed only during addition and deletion of strings in the set, and these updates are relatively less frequent than the actual query process itself, the authors suggest the CBF should be maintained in software and the corresponding bit vector in hardware, thus saving memory resources.

However, several research efforts [4, 5] show how to evade standard pattern matching techniques by splitting into several packets or by changing (e.g., by UTF-8 synonyms) the malicious strings slightly, thus making useless the pattern matching on single packets. Many software tools (e.g., FTester, FragRoute, or Nikto) even implement such evasion attacks. This work focuses on the "fragmentation" evasion (throughout the article we use the term *evasion* to refer to it). Currently, the only way to deal with this problem is to reassemble the overall packet flows and afterward apply standard pattern-matching algorithms. This dramatically increases requirements for security systems in terms of both memory and processing power, especially for securing traffic at wire speed. Moreover, to face a few malicious flows, an IDS must reassemble all passing flows. In [4], it is shown that the processing for TCP reassem-

0890-8044/09/$25.00 © 2009 IEEE

bly can be reduced remarkably by optimizing for the expected case when most TCP segments are in order. However, the costs for both memory and processing remain too high.

Some work tries to avoid the requirement for flow reassembly to detect evasion attacks. The authors of [6] propose an architecture composed of a flow processor and a payload processor. The former maintains per flow state information for multi-packet signature detection, whereas the latter uses a combination of parallel BFs. More precisely, the payload processor adopts, for each length, a BF that represents all the strings of that length, as well as a BF that represents all the string pieces of that length. When a packet arrives, a complete check is performed on all the filters (an expensive process). If a match is detected, the flow database is updated, and the state becomes malicious (if a whole signature is found) or suspicious (if a simple piece is found). Whenever the flow state is malicious, the flow is passed to an analyzer for a further deterministic check. This scheme assumes that packets are not ambiguous, in order, and not overlapped, thus neglecting many real issues. Moreover, the use of filters for prefixes of one or two bytes appears too expensive for memory requirements, processing power, and alert rate, thus making such a system inefficient.

The basic idea of [7] is to split the signatures to be searched by pattern-matching algorithms into small substrings. In this way, if a sufficiently large piece is completely inserted in a packet, it is easily detected. Otherwise, the attacker is forced to use several very small or out-of-order packets, and such abnormal behaviors are revealed by adopting proper heuristics. Both techniques are performed in the fast data path, thus guaranteeing a big saving in terms of both time and memory with respect to the overall flow reassembly. However, this solution presents some weaknesses: when the counter of small or out-of-order packets of a specific flow exceeds a threshold, such a flow is diverted to the slow data path. The article claims that this threshold is set according to the signature length that the small packets belong to. Unfortunately, it is not possible to know such a parameter before the flow has been reassembled and the entire signature has been detected; hence, this heuristic appears difficult to use.

Anti-evasion is a difficult problem and to find a conclusive solution is very difficult. Hence, our work is an attempt to address the problem in an alternate and effective way, thus creating new opportunities for future research on this topic. The main goal is the same as that of [6] and [7]: avoid flow reassembly for detecting an evasion attack. First, we show how CBFs can be used for anti-evasion techniques due to their appealing features in terms of compactness and speed, update capability, and emptying feasibility. Then, a comprehensive CBF-based solution for anti-evasion is illustrated. More precisely, the article is organized as follows. The next section illustrates the usefulness of CBFs in pattern matching and anti-evasion. Then, we describe the main concepts of our system, its overall functioning, and the heuristics used for specific patterns of attacks. Finally, we present some potential improvements of our solution, and an experimental evaluation concludes the article.

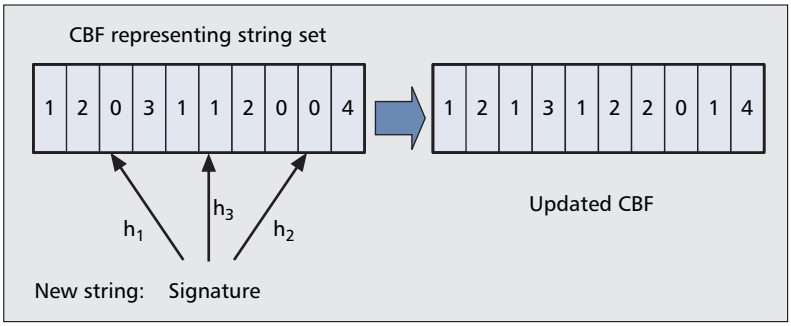


Figure 1. *Addition of a new string in a CBF.*

CBFs for Pattern Matching and Anti-Evasion

A Bloom filter represents a set S of n elements from a universe U by using an array of m bits, denoted as $B[1], \ldots, B[m]$, initially all set to 0. The filter uses k independent hash functions $h_1, \ldots, h_k$ with $\log_2(m)$ bits long output, which independently map each element in the universe to a random number uniformly over the range. For each element x in S, the bits $B[h_i(x)]$ are set to 1 for $1 \le i \le k$ (a bit can be set to 1 multiple times).

To answer a query of the form "Is y in S?", one checks whether all $B[h_i(y)]$ are set to 1. If not, y is not a member of S, by construction. If all $B[h_i(y)]$ are set to 1, it is assumed that y is in S; hence, a BF may yield a false positive. The probability of a false positive (f) can be tuned opportunely by choosing the proper values for m and k.

However, BFs do not allow changes in the item set. In fact, deletion cannot be accomplished simply by changing ones into zeros, as a single bit may correspond to multiple elements. Therefore, CBFs were introduced, which are based on the same idea of BFs, but use fixed-size counters (also called bins) instead of single bits of presence. When an item is inserted, the corresponding counters are increased; then, deletions can be performed safely by decreasing the counters.

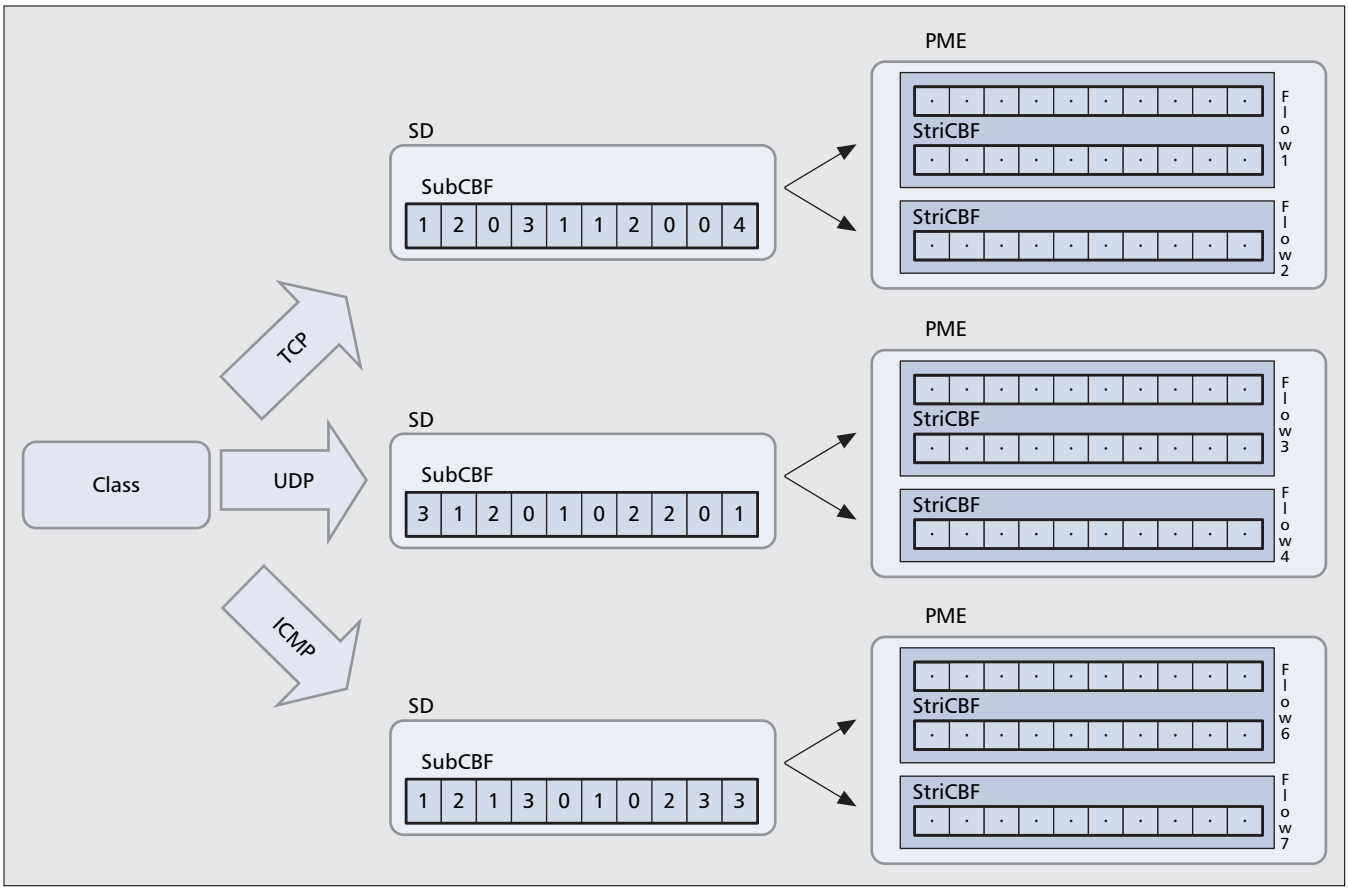
CBFs are used in representing elements for their well-known compactness and speed. Our idea is also to use them for pattern matching and anti-evasion due to their innovative capabilities of quickly updating the set they represent and counting the occurrences of elements. The first property can be used to rapidly take into account each new virus definition, with no need to rebuild the overall structure: to add a new string to be matched, it is sufficient to apply the hash functions to such a string and increase the proper CBF bins (as shown in Fig. 1, where the new string SIGNATURE is added and h_i are the hash functions). The same principles can be used for removing an obsolete string.

Counting the occurrences of elements makes CBFs appealing just for anti-evasion. Indeed, a CBF can be set to represent the different substrings composing a string: the arrival of any pieces belonging to the string triggers a decrease of the proper bins and when the filter is completely reset to zero, the overall match is detected. In this way, to reveal an evasion attack, it is no longer necessary to divert a flow to a slow data path engine, which must reassemble the flow and perform pattern matching. The detection, unlike when using BFs, can be performed completely in the fast data path, thus speeding up the overall performance of an IDS.

The Anti-Evasion System

Motivations and Ideas

The main idea of our system is to split a priori the strings to be searched in three-byte-long substrings and create a CBF representing them (hereafter called "substring CBF" or simply subCBF) for preliminary pattern matching. When a substring is detected through the subCBF, a bank of further filters (called string CBFs, or striCBFs) is properly set for the specific flow: more precisely, a filter is initialized for each string that the detected substring belongs to. In this way, all the next packets of that flow are processed in search of the remaining characters of the strings: whenever a striCBF is completely reset to zero, the attack is detected and the flow is blocked.



■ Figure 2. *The scheme of our system.*

However, not all the attacks can be detected in this way; for instance, a string split in several very small packets (less than three bytes) is not revealed because the substring detectors search only for substrings of three bytes. Therefore, we plan to divert such packets (very infrequent in real traffic) to the slow data path for flow reassembly and pattern matching. Moreover we set a threshold on the maximum number of flows to be diverted, thus avoiding denial of service attacks.

The only assumption we make is that packets entering our anti-evasion system are not ambiguous, that is, packets do not overlap. To force this condition, we assume a traffic normalizer (like the one described in [8]) before our system, which keeps traffic flows consistent and solves any ambiguities. The arrival of out-of-order packets does not affect the correct functioning of the system and the proper detection of attacks.

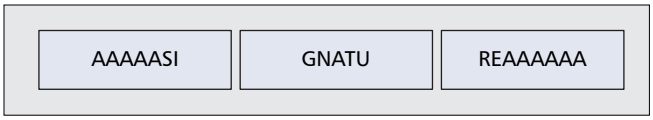
System Architecture

Our architecture, shown in Fig. 2, is composed of several modules. At first, traffic flows are divided by a classifier according to transport protocols and forwarded to different engines, named substring detectors (SDs). Such a first division enables the balancing of the load among the SDs and decreases the size of their relative filters. Each SD performs a pattern matching on the overall content of packets by using its specific subCBF, which represents the set of strings that identifies the valid attacks for that protocol. More precisely, a subCBF represents substrings of three bytes; such a specific length has been selected also to reveal the shortest strings, which are six bytes long, as pointed out from the analysis of SNORT data sets.

Consequently, a substring detector processes all the ingoing traffic of a specific transport protocol, by moving along an inspection window of three bytes. When all the hash functions applied to a group of three bytes point to full bins of the subCBF, the SD determines that the substring was detected. As previously mentioned, the use of CBFs in this phase allows for a fast update of a string set. In particular, we choose multilayered compressed counting Bloom filters [9] to implement subCBFs: the first layer, which is used for the frequent lookups, can be placed in a fast and small memory, whereas the other layers, useful for string-set updates, can be stored in a slower memory.

After a malicious substring is detected by one of the SDs, a block of striCBFs are set to determine if such an alert actually corresponds to a complete attack. These filters are built a priori and stored in memory: they are addressed by simple hash tables, which in turn are efficiently indexed by the bins of the subCBFs (as suggested in [10]). Then, the striCBFs are handled by other modules, the so-called pattern-matching engines (PMEs), which process the specific traffic flow that the classification stage forwards to it (by means of a classification rule that is set by the SD after the substring detection). The goal of each PME is to perform an overall pattern matching on the flow to determine if the detected substring is actually a piece of a string or simply a false positive. For this purpose, the PME sets a striCBF for each string that the detected substring belongs to. Such a CBF represents all the remaining characters of the string in the format (*char*, *pos*), where *pos* is, for example, the TCP sequence number of the character *char*. More precisely, the sequence number of the byte that gets off the string is associated to the filter, so *pos* is actually the relative position with respect to such a value. Whenever a striCBF is completely reset to zero, it is assumed that the string was detected, and the packets must be dropped to nullify the attack.

Furthermore, the string length also is associated to the fil-



■ Figure 3. *The string is SIGNATURE. The piece SI is not outright detected, and when the piece GNA generates an alert, it has already been processed.*

ter; from such a value and from the beginning point of the string, the engine can understand which bytes of the flow it must process and which bytes cannot belong to the string. In addition, by processing the proper fields of the TCP header, the engine can determine when the filter must be removed because it "has expired."

Clearly, the bins corresponding to the characters of the detected substring that started the filter are decreased in the initialization phase. In this way, the functioning of PMEs is independent of the arrival order of packets: also a "middle" substring that arrives as first leads off the striCBF setting, and this does not affect the correct functioning of the system.

Small Packets

If an attacker splits the signature in several one- or two-byte long packets (hereafter, we call them small packets), the system cannot detect the attack because the substring detectors search for substrings of three bytes. Fortunately, packets of one to two bytes are very rare in real traffic,[1] except for certain applications such as telnet and ssh; therefore, we can use their presence as an alert and adopt proper expedients.

In previous work on anti-evasion systems, this type of attack was faced only partially. In [6], a prefix Bloom filter is set for each substring length, but filters representing pieces of one or two bytes require an excessive amount of memory; moreover, they trigger a hard processing for each packet and an intolerable alert rate (each time a character belonging to a string is found, an alert is generated). Instead, the authors of [7] propose a heuristic that is very difficult to apply in practice, as illustrated in the first section.

Our idea to thwart this type of attack is to divert all the small packets to a slow path engine; this policy is based on the consideration that they are very infrequent in real traffic. The slow path engine must reassemble such flows and perform a deterministic pattern matching on them to verify the actual presence of an attack. To face denial of service attacks, we also select a threshold on the maximum number of flows to be diverted. Otherwise, attackers could inject a small packet for each flow and force the system to divert and reassemble all the flows.

System Optimization

Our system can be improved, in terms of both functioning and performance, by adopting a series of refinements.

Analyzing real data sets of malicious signatures, we noticed the presence of a few substrings that are very frequent, both in the malicious strings and in normal traffic. We plan to delete such substrings from the filters, thus saving memory (smaller filters) and processing load (less substrings that generate an alert). What are the potential drawbacks? Deleting such frequent strings from the subCBFs could result in a lower detection capability because fewer substrings signal an attack. Furthermore, the deletion from the striCBFs could increase false positives because filters can be more easily reset to zero. However, the experimental results in the next section show just a slight increase of detection capability and decrease of false positives, hence the adoption of such an expedient turns out to be convenient.

Our standard system does not detect all possible patterns of attacks. For example, in some cases the beginning of a signature is missed, because it is too short to be revealed by the substring detectors, whereas it belongs to a packet that is too long to be classified as small (as *SI* in Fig. 3). Therefore, the packet is identified as normal and whenever another fragment (*GNATU* in the figure) triggers a striCBF setting, this piece already has been processed. In this way, some bins continue to be full, and the filter is never completely reset to zero. Moreover, to speed up processing, one might consider not waiting for the overall emptying of filters. Therefore, for the efficiency of our system, it seems advisable to set an "emptying threshold" α for the striCBFs: when α is exceeded, the attack is considered as detected. Such a threshold is computed as the ratio between the number of bin decrements and the sum of all the bin counters, where α equal to 1 means that the overall filter must be depleted, whereas lowering the threshold results in a faster detection. It is necessary to find the correct tradeoff between a higher speed and a larger number of potential false positives (as shown in the next section).

With respect to the memory efficiency, whereas CBFs (or their improved versions, multilayer compressed counting Bloom filters [ML-CCBF] [9]), blooming trees [11], or dl-CBF [12]) are the best choices for subCBFs, striCBFs may not require the minimum amount of memory for their function, that is, to recognize simple characters. Thus, striCBFs can be replaced by the actual signature string and a bitmap (one bit per signature byte) that indicates whether each character has been found or not; by pursuing this approach, we use nine bits per signature byte. Instead, with a plain CBF, we would need a number of bits per character equal to $4k/\ln 2$ ($k = \log 2f$ where f is the false positive probability), which falls down approximately to $k/\ln 2$ bits if a blooming tree (BT) is used. Of course when k is large, this amount can be larger than nine bits.

Therefore, the choice of using either the string itself or a CBF (or, better, a BT) is related to the parameter f, which is, in turn, computed according to the total number of times n_q we query that particular filter. In fact, the mean number of false positives is given by the product of f and n_q; as we do not have overlapping packets (because of the normalizer), the number of times n_q we check a given striCBF that represents a n-bytes string is exactly n. Then, by simply selecting $f \times n = 2^{-k} \times n \ll 1$, we can cope with false positives largely by limiting their mean number. In practice, we have found that when $n < 10$ (remember that striCBFs do not include the three-byte substrings found by subCBFs, thus further reducing n), we can achieve to use less than nine bits per character without increasing the amount of false positives. In the experiments shown in the next section, we used the combinations of BTs and strings+bitmaps that minimize the amount of memory used.

Experimental Results

For the experimental runs, a cluster of PCs that generate traffic toward a LAN is used: one of them runs FTester, which is a software tool designed for testing IDS capabilities, while the other ones generate background traffic. A general purpose PC running our anti-evasion system is placed before the LAN to protect it. We are not required to use a normalizer (included by our system) because we set FTester to generate attacks with no ambiguities.

FTester can create evasion attacks, by splitting signatures among several packets and with different lengths and number

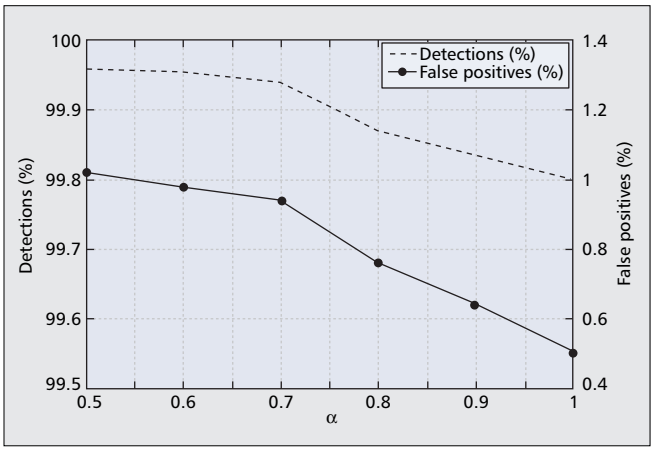
[1] *As shown for example by data at http://netflow.internet2.edu/*

Trace	Size (Mbytes)	Normal attacks			Small attacks			Total attacks		
		Generated	Detected	False pos.	Generated	Detected	False pos.	Generated	Detected	False pos.
Tr1	224	4240	99.8%	0.23%	312	100%	4.8%	4552	99.8%	0.5%
Tr2	190	3800	98.9%	0.1%	310	100%	4.8%	4110	99%	0.46%
Tr3	160	1418	99.9%	0.35%	200	100%	3.5%	1618	99.9%	0.74%
Tr4	100	1213	100%	0.32%	185	100%	4.3%	1398	100%	0.85%
Tr5	50	789	99.2%	0.25%	108	100%	6.4%	897	99.2%	1%

■ Table 1. *Performance of the standard system in terms of detected attacks and false positives..*

Trace	Detected att. (%)		False pos. (%)	
	NFS	ST	NFS	ST
Tr1	99.7	99.8	0.26	0.23
Tr2	98.9	98.9	0.13	0.1
Tr3	99.8	99.8	0.35	0.35
Tr4	99.8	100	0.41	0.32
Tr5	99.2	99.2	0.5	0.25

■ Table 2. *The effects of deleting the most frequent substrings.*



■ Figure 4. *Detection percentage and false positives by varying α.*

of signatures, order of pieces, and so on. In particular, we use the following options of FTester (let us suppose the malicious string is ATTACK):

- *--e stream* (simple splitting of the TCP stream): packet = [packet1(ATT) + packet2(ACK)]
- *--e frag*1 (out of order packets): packet = [fragment3(C) + fragment2(TA) + fragment1(AT) + fragment4(K)]
- *--e frag*2 (like frag1 but send the last fragment first): packet = [fragment4(K) + fragment3(C) + fragment2(TA) + fragment1(AT)]

The lengths of substrings in our runs are alternated to have both "normal" evasion attacks (with substrings of almost three bytes) and attacks with small packets (less than two bytes). In Table 1, for traces of different sizes and by distinguishing between normal and small attacks, the following data are reported:

- The number of attacks that are generated
- The percentage of real attacks we detect
- The percentage of false positives

The last three columns enumerate the total values. Such measurements refer to the standard functioning of the system; the effects of the optimizations listed in the previous section are shown later.

These results exhibit high percentages of detection, whereas the number of false positives remains small. As foreseen, the technique used against the attacks performed with small packets generates the largest number of false alerts because each small packet is signaled as a potential attack.

In Table 2, the potential drawbacks of deleting off-line the most frequent substrings from the filters are shown. The traces under test are the same as in Table 1, and we refer only to the normal attacks (i.e., columns 3–5 in Table 1) because the small attacks are not affected by such a modification. The number of detected attacks by deleting such substrings ("No Frequent Substrings," NFS in the table) is practically equal to those revealed by the standard system (ST), whereas the number of false positives increases slightly. These results justify the adoption of such a refinement, which without remarkable additional cost, improves the efficiency of the system in terms of both memory and speed. Specifically, by adopting such an expedient, we observe a mean reduction of memory footprint by a factor of four in the experiments, thus requiring less than 100 bytes for each flow processed by the PMEs.

Finally, for the processing of the trace Tr1, Fig. 4 shows the effects of the "emptying threshold" α that enables us to detect a string even though the relative striCBF was not reset to zero completely. It is clear that choosing low values of α results in a higher percentage of detection but also in a larger number of false positives, whereas for high values, the opposite happens. In any case, the number of detected attacks is sufficiently high (i.e., beyond 99 percent).

Conclusions

This article presents a CBF-based system for anti-evasion. CBFs are very appealing structures for this purpose because they efficiently represent elements, perform fast lookup, quickly update the string sets, and enable detecting a split signature.

In detail, in our system the traffic flows are divided by a classifier according to the transport protocol and forwarded to the substring detectors, which reveal the presence of dangerous substrings; such a preliminary detection enables a pattern-matching engine to search for the remaining characters of the malicious string.

The improvement of our proposal is that all the traffic is

processed in the fast data path, thus achieving a large increase in speed. Indeed, by using the CBF, we can perform an overall string matching, with no need to reassemble the flow in the slow data path; the cost is a given probability of false positives, which is inherent in CBFs and can be properly tuned when setting the filters. For very small packets only (shorter than three bytes), we are forced to divert the flows to the slow data path. A series of refinements were presented to solve some issues or improve system performance.

As for results, this solution achieves a detection rate of nearly 99 percent. We plan to perform more accurate measurements to illustrate the advantages in terms of processing speed and to determine the maximum number of attacks per second that can be revealed and the memory consumption per flow.

Acknowledgments

This work was partially sponsored by the European Project FP7-ICT PRISM, contract number 215350 and by the "Fondazione Cassa di Risparmio di Pisa" research program FRINP.

References

[1] S. Dharmapurikar *et al.*, "Deep Packet Inspection Using Parallel Bloom Filters," *IEEE Micro*, vol. 24, no. 1, 2004, pp. 52–61.
[2] M. Nourani and P. Katta, "Bloom Filter Accelerator for String Matching," *Proc. 16th Int'l. Conf. Comp. Commun. Net.*, pp. 185–90.
[3] L. Fan et al., "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *SIGCOMM Comp. Commun. Rev.*, vol. 28, no. 4, 1998, pp. 254–65.
[4] T. H. Ptacek and T. N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," *Secure Networks, Inc., T2R-0Y6, Tech. Rep.*, Alberta, Canada, 1998; http://citeseer.ist.psu.edu/ ptacek98insertion.html
[5] M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," *Proc. 10th Conf. USENIX Sec. Symp.*, 2001, p. 9.
[6] N. S. Artan and H. J. Chao, "Multi-Packet Signature Detection Using Prefix Bloom Filters," *Proc. IEEE GLOBECOM*, vol. 3, 2005, pp. 1811–16.
[7] G. Varghese, J. A. Fingerhut, and F. Bonomi, "Detecting Evasion Attacks at High Speeds without Reassembly," *SIGCOMM Comp. Commun. Rev.*, vol. 36, no. 4, 2006, pp. 327–38.
[8] M. Vutukuru, H. Balakrishnan, and V. Paxson, "Efficient and Robust TCP Stream Normalization," *IEEE Symp. Sec. Privacy*, Oakland, CA, May 2008.
[9] D. Ficara *et al.*, "Multilayer Compressed Counting Bloom Filters," *Proc. INFOCOM '08*, Apr. 2008.
[10] H. Song *et al.*, "Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing," *Proc. 2005 Conf. Apps., Tech., Architectures, Protocols Comp. Commun.*, New York, NY, 2005, pp. 181–92.
[11] D. Ficara *et al.*, "Blooming Trees: Space-Efficient Structures for Data Representation," *Proc. IEEE ICC '08*, May 2008.
[12] F. Bonomi *et al.*, "An Improved Construction for Counting Bloom Filters," *LNCS 4168, 14th Annual Euro.Symp. Algorithms*, 2006, pp. 684–95.

Biographies

GIANNI ANTICHI (fgianni.antichi@iet.unipi.it) received his Master's degree in telecommunication engineering in September 2007 from the University of Pisa, by discussing a thesis on implementation of a high-performance IP traffic generator. In January 2008 he entered, as a Ph.D. student, the Department of Information Engineering at the University of Pisa, where he is currently doing research in the areas of packet classification, network processors, and FPGA

DOMENICO FICARA (domenico.ficara@iet.unipi.it) received a Master's degree in telecommunication engineering from the University of Pisa in 2005, discussing a thesis on resource scheduling in network processors. In January 2007 he joined the Department of Information Engineering at the University of Pisa as a Ph.D. student, where he is currently doing research in the area of network algorithmics, deep packet inspection, network tomography, and topology discovery.

STEFANO GIORDANO (stefano.giordano@iet.unipi.it) received a Laurea degree in electronics engineering and a Ph.D. degree in information engineering, both from the University of Pisa. He is an associate professor with the Department of Information Engineering of the University of Pisa. He is a referee for the European Union, the NSF, and the Italian MIUR and MAP Ministries. His research interests are telecommunication networks analysis and design, simulation of communication networks, and multimedia communications.

GREGORIO PROCISSI (gregorio.procissi@iet.unipi.it) received a graduate degree in telecommunication engineering in 1997 and a Ph.D. degree in information engineering in 2002 from the University of Pisa. From 2000 to 2001 he was a visiting scholar at the Computer Science Department of University of California at Los Angeles. In September 2002 he became a researcher with CNIT. Since 2005 he has been an assistant professor with the Department of Information Engineering of the University of Pisa. His research interests are measurements and performance evaluation of IP networks.

FABIO VITUCCI (fabio.vituccig@iet.unipi.it) received, from the University of Pisa, his Master's degree in telecommunication engineering in October 2004 with a thesis on simulative analysis of MPLS networks, and a Ph.D. degree in information engineering in June 2008 with a thesis on network processors. He is currently doing research at the Department of Information Engineering of the University of Pisa in the areas of packet classification, pattern matching, and network processors.