Packet Chunking for File Detection

Junnyung Hur, Hyeon Gy Shon, Young Jae Kim, and Myungkeun Yoon, *Member, IEEE*

***Abstract*— Network-based intrusion detection and data leakage prevention systems inspect packets to detect if critical files such as malware or confidential documents are transferred. However, this kind of detection requires heavy computing resources in reassembling packets and only well-known protocols can be interpreted. Besides, finding similar files from a storage requires pairwise comparisons. In this paper, we present a new network-based file identification scheme that inspects packets independently without reassembly and finds similar files through inverted indexing instead of pairwise comparison. We use a content-based chunking algorithm to consistently divide both files and packets into multiple byte sequences, called chunks. If a packet is a part of a file, they would have common chunks. The challenging problem is that packet chunking and inverted-index search should be fast and scalable enough for packet processing. The file identification should be accurate although many chunks are noises. In this paper, we use a small Bloom filter and a two-level threshold strategy to solve the problems. To the best of our knowledge, this is the first scheme that identifies a specific critical file from a packet over unknown protocols. Experimental results show that the proposed scheme can successfully identify a critical file from a packet without packet reassembly.**

***Index Terms*— Bloom filter, data leakage prevention, inverted index, cybersecurity, content-based chunking.**

I. Introduction

NETWORK-BASED intrusion detection systems (IDS) and data leakage prevention (DLP) systems monitor network traffic and inspect packets. They detect if critical files such as malware downloads or confidential documents are transferred over the network [2], [3]. In this paper, we call those files that are not supposed to be transferred over the network by security policies as critical files. Critical files can be identified at host level or network level. In this paper, we focus on network-based detection of critical files by packet inspection.

Manuscript received 1 March 2022; revised 30 August 2022; accepted 23 September 2022; approved by IEEE/ACM Transactions on Networking Editor J. S. Sun. Date of publication 31 October 2022; date of current version 16 June 2023. This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government through the Ministry of Science and Infomation and communiactions Technology (MSIT) under Grant NRF-2020R1A2C1006135; in part by the Institute for Information and Communications Technology Promotion (IITP) Grant funded by the Korean Government through MSIT (fundamental research on real-time similarity measurement for automatic analysis of big data in security) under Grant 2018-0-00429; and in part by MSIT, South Korea, under the ICT Challenge and Advanced Network of Human Resources Development (HRD) (ICAN) Program supervised by the Institute of Information and Communications Technology Planning and Evaluation (IITP) under Grant IITP-2020-0-01826. Part of the results was presented in INFOCOM 2021 [1]. *(Corresponding author: Myungkeun Yoon.)*

Junnyung Hur and Myungkeun Yoon are with the Department of Computer Science, Kookmin University, Seoul 02707, South Korea (e-mail: mkyoon@kookmin.ac.kr).

Hyeon Gy Shon is with MUHAYU, Seoul 04793, South Korea.

Young Jae Kim is with Common Computer, Seoul 06770, South Korea.

Digital Object Identifier 10.1109/TNET.2022.3215549

Deep packet inspection has been extensively studied to monitor network traffic and analyze packets in details, not only header information but also payload that may include partial contents of a critical file. Encrypted packets are beyond the scope of this paper because they cannot be interpreted properly. Although the amount of encrypted traffic increases, recent studies show that non-negligible amounts of packets are still not encrypted [2], [3]. For example, inadvertent employees may send a confidential file to themselves by non-encrypted emails. Network-based deep packet inspection devices can detect this potential data leakage by inspecting packets.

We observe that not only critical files but also their similar files should be detected when they are transferred over the network. For example, a previous version of a confidential document may contain the same critical information, but employees may transfer them as plain texts. A recent study also shows that malware variants can have similar byte streams except a few different bytes [4]. Therefore, we can detect a variant malware file by using its similar file. In this paper, we say two files are similar if their byte sequences are similar. Finding similar files is a much more difficult problem than finding exact files. Two identical files have the same hash value, which makes the problem easy. However, similar files have totally different hash values. Special fuzzy hashes are developed to find similar files [5], but fuzzy hash values should be compared in pairwise to measure the similarity between two files. If there are many files to compare, the fuzzy hash becomes inefficient in finding similar files.

To find a critical file that is similar to a transferred file, the current best practice first reassembles all related packets, and then interprets the network protocol to extract an embedded file [6], [7]. If a file is extracted successfully from the packets, a fuzzy hash value of the file is computed. Finally, the fuzzy-hash value is compared with all the fuzzy-hash values of critical files in pairwise. Each pairwise comparison produces a similarity score, and the critical file of the highest score is selected. This practice has many limitations; first, reassembling packets consumes computing resources heavily. Second, only well-known protocols can be properly interpreted. Third, a complete file is not restored if at least one packet is missing during the reassembly. Fourth, a fuzzy hash operation requires inefficient pairwise comparisons.

In this paper, we present a new critical file identification scheme that for a given packet or a group of packets, a specific critical file should be identified if the file has significant amounts of common contents with the packet payloads. The new scheme uses a content-based chunking algorithm to consistently divide both files and packet payloads into multiple byte sequences, called chunks [8], [9]. If a packet payload is a

1558-2566 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 1

part of a file, the packet and the file would have many chunks in common. We build an inverted index from the collection of all critical files [10], [11].

The challenging problem is that packet chunking and inverted-index search should be fast and scalable to handle an enormous number of packets. The file identification should be accurate although most chunks are just a noise. In this paper, we use a small Bloom filter [12], [13] and a two-level threshold strategy to solve the challenging problems. To the best of our knowledge, this is the first scheme that identifies a specific critical file from a packet or a group of packets over even unknown protocols without packet reassembly. Experimental results show that the proposed scheme can successfully identify a critical file from packets.

The rest of the paper is organized as follows. We introduce the problem and motivation in Section II. We present the new scheme in Section III and the experimental results in Section IV. Sections V and VI are related work and conclusions.

II. Problem and Motivation

We define critical files as confidential documents or malware files that are not supposed to be transferred over network packets. Critical files are assumed to be already collected in a database for an enterprise or an institution. In this paper, we consider file similarity at byte-sequence level. If two files include many byte sequences in common, these two files are similar to each other. We focus on network-based detection of critical files or their similar versions through deep packet inspection while these files are transferred in multiple packets.

Similar files are common in confidential documents and malware files; first, a confidential document file is generally similar to its previous version if little contents are updated. Therefore, the previous version should be identified carefully because it may have critical overlapping contents in common. In many cases, only the final version is kept confidentially while old versions may be inadvertently left in even personal devices.

Second, a recent study shows that there are a large amount of malware files that are similar to each other; only a small number of bytes are different among the files of malware variants [4]. In this paper, we are not interested in malware files of semantically similar behaviors. We focus on only those malware files that look similar at byte-sequence level. Therefore, we may identify two installer-type malware files as similar if most parts of the byte-sequences look same. In this case, malware analysts need to further review the similar files manually. In terms of cybersecurity, we reveal that finding similar files cannot help identify new types of malware; this only helps identify malware variants whose files are similar to a known malware file at byte-sequence level [14], [15], [16].

Similar files cannot be identified by simple signature of cryptographic hash. One-byte difference would generate totally different file signature values. To compare similar files, a number of fuzzy hashing has been proposed [5], [17]. For example,

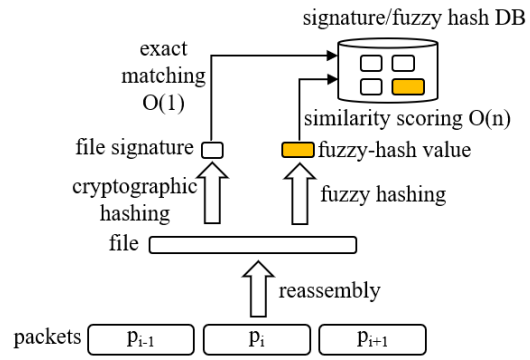

Fig. 1. Current best practice for detecting identical files and similar files by cryptographic hash and fuzzy hash.

we explain ssdeep, a de facto standard for fuzzy hashing; the idea is that a file is divided into multiple non-overlapping fragments and each fragment is hashed. Then, small bits from each hashed value is extracted and concatenated to build a representative string for the whole file. If two files are similar, their representative strings are close in terms of such measurement metrics as edit distance, jaccard index, cosine similarity, etc. [18]. However, a challenging problem is that the representative strings should be compared in pairwise. This causes $O(n)$ time complexity to find similar cousins for a given file where n is the number of critical files. Fuzzy hashing cannot compare a large number of files scalably.

We study the problem that, for a given packet or a group of packets, a specific critical file should be identified if the file has significant amounts of common contents with the packet payloads. This is a more challenging problem than comparing two similar files; first, the byte sequence of a packet can change even if the same file is transferred. For example, network protocols exchange meta information such as time, version, checksum while the byte sequence of the file is divided into multiple network payloads; if the size of meta information changes, byte sequences of all packets are affected, known as a boundaries-shift problem [8], [19]. Second, current best practices rely on packet reassembly to extract an transferred file first, and then the file is used to find its similar critical files [6], [7], [20]. This works for only a small number of well-known protocols that can be properly parsed and interpreted. If a file is transferred on unknown protocols, this scheme fails. Fig. 1 shows the current best practice that finds the same file by cryptographic hashing and the similar file by fuzzy hashing. Third, any file similar to a critical file can be transferred that should be detected. Some parts of packets are equal to some fragments of the critical file. However, other parts can be totally different, and we do not know which parts would be same or not. Last but not least, network processing should be fast enough to support line speed. In the literature of network-based packet inspection such as intrusion detection systems and firewalls, it is well known that aggregating packets of the same session consumes computing resources heavily. Therefore, the number of sessions and packets for inspection should be minimized and packet reassembly is not preferred. We observe that one packet or a small group of packets have decisive information in

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 2

identifying a critical file. An Ethernet packet can have more than 1,400 bytes for payload that includes some parts of a transferred file. Besides, a file is generally transferred over a sequence of contiguous packets rather than interleaved ones. Therefore, if we extract and aggregate meaningful information from a packet or a series of packets, we would obtain an important hint to determine which critical file is similar to the transferred file. However, the challenging problems described above should be cleared first. In this paper, we present a new scheme to solve the problems.

III. Packet Chunking for File Detection

We present a fast and scalable detection scheme to identify a critical file that has common contents with a given packet or a group of packets. The proposed scheme builds an inverted index for the collection of critical files after each file is divided into content-based chunks. For a given packet, the same chunking algorithm extracts chunks from the packet, and the chunks are used as a query to search the inverted index for the most relevant critical file. Because the search takes time, we use a bloom filter to select only those packets that deserve the search. We explain first how to build the inverted index for a given collection of critical files through a content-based chunking (CBC) algorithm, and then how to inspect a packet to find the most relevant critical file over the inverted index. Finally, we present how to optimize the query and search. The proposed scheme is implemented on Elasticsearch, the world's leading open source search and analytics solution [11].

A. Content-Based Chunking and Inverted Index

We assume that critical files are collected and the collection is denoted as $F = \{f_1, f_2, \ldots, f_n\}$, where f_i is the ith file. We use a CBC algorithm to divide the byte sequence of a file into multiple non-overlapping chunks. In this paper, we use AE chunking algorithm that outperforms traditional rolling hash algorithms [8], but any chunking algorithm can be used instead. Reading byte streams from a file, AE generates anchors based on local contents of the file. An anchor position is computed from a local maximum in AE chunking, and the comparison window is dynamically expanded depending on local byte streams within the window. An anchor is a position where to divide the file. The byte sequence between two consecutive anchors is called a chunk, and we adjust the parameters of the AE algorithm to generate chunks whose average size becomes w bytes. We use $f_{i,j}$ to denote the jth chunk of the ith file in the critical file collection. Readers are recommended to read [8] for more details about AE chunking. Fig. 2 shows how file f_i is divided into six file chunks of $[f_{i,0},\ldots,f_{i,5}]$.

After each file is divided into multiple content-based chunks, we build an inverted index from hashed chunks to files [10]. We use $h(f_{i,j})$ to denote the hashed value of chunk $f_{i,j}$. Fig. 3 shows how an inverted index is built from critical files after they are divided into content-based chunks. We use each hashed chunk as a search key. When a set of hashed chunks is given as a query set, we can look up the inverted index to select the most relevant critical file depending on the scoring algorithm. The scoring algorithm gives each key a weight, and the total score of each critical file is computed on the weights. For example, we implement our proposed scheme on the Elasticsearch platform. The default scoring algorithm of Elasticsearch is Okapi BM25 that is basically an advanced scheme of traditional term-frequency/inverse document frequency (TF-IDF) [10], [11].

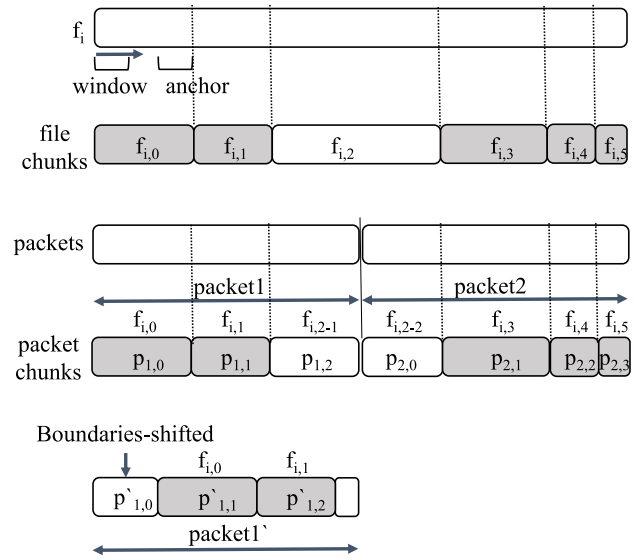

Fig. 2. File-level chunking vs packet-level chunking.

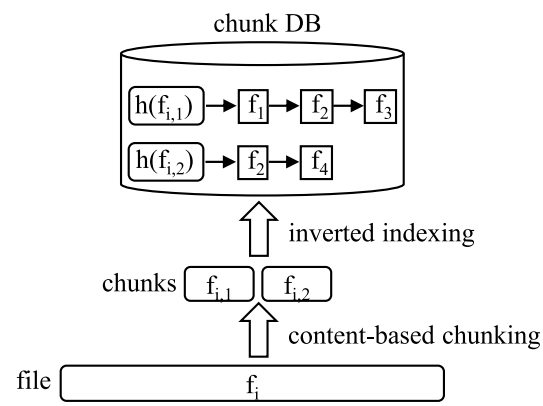

Fig. 3. Inverted index for the collection of critical files where hashed chunks are keys.

In this paper, we tune the scoring algorithm of the inverted index; for a given query set of hashed chunks, the scoring algorithm prefers a critical file that has a larger number of elements from the query set. In other words, a critical file is returned as a search result if the critical file includes the largest number of query set elements.

B. Packet Inspection and Naive Search

We assume that a packet monitoring device is deployed and every packet can be provided for inspection. In general, the monitoring device is installed on the network perimeters to

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.

inspect incoming and outgoing packets between internal and external networks. Recently, network security devices such as firewalls, intrusion detection systems, often provide a packet capturing service. If the speed of a network line is high, it is non-trivial to capture all packets without any loss. There are a number of research work and practices about packet capturing, which is beyond the scope of this paper. We assume that captured packets are stably provided.

For a given packet, we apply the same AE chunking algorithm with the same average chunking size of w that was used for building the inverted index from critical files. A set of content-based chunks are generated for every packet. If we use this set of chunks per packet as a query set and search the inverted index for relevant critical files, candidate files would be retrieved. However, this naive scheme is not feasible for two reasons: first, the search cannot be repeated per packet because there are huge amounts of packets. We call this a scalability problem. Second, a critical file is often retrieved falsely that actually does not share much common contents with the packet. We call this a false-positive problem. This happens when the packet contains contents that appear in not only the critical files but also other ordinary files.

Fig. 4 shows this naive search where two chunks of $p_{i,1}$ and $p_{i,2}$ are extracted from packet p_i. When the query set of $\{h(p_{i,1}), h(p_{i,2})\}$ is received, the inverted index retrieves f_2 as the most relevant search result. Note that only f_2 has both elements of the query set and appears in both lists of keys $h(p_{i,1})$ and $h(p_{i,2})$; the other critical files have only one element. The notation of a chunk can be used interchangeably as $p_{i,j}$ or $f_{x,y}$. $p_{i,j}$ represents a chunk from a packet while $f_{x,y}$ is from a file. In this vein, we search the inverted index for a critical file that has as many $f_{x,y} = p_{i,j}$ cases as possible.

The search quality depends on how many identical chunks are generated from a critical file and a packet when the packet belongs to the transferred critical file or its similar one. We argue that most of the chunks generated from the packet will overlap those of the critical file if the average chunk size of w is properly set. This is illustrated in the middle plot of Fig. 2 where most of the chunks from two packets preserve their byte sequences and there are 5 common chunks between file chunks and packet chunks. In this figure, the file is divided into two packets. The first packet generates two same chunks and the second packet generates three same chunks, compared with file chunks. In general, a packet would generate the same chunks except the first and last chunks. Some packets would generate more different chunks, but we confirm from experiments that the CBC algorithm recovers the boundaries-shift problem quickly after 1∼3 chunks are generated.

At the bottom of Fig. 2, packet1' includes extra appending bytes to packet1, for example variable-sized header information of an application payload, and they shifts the packet content to the right from the left-most side. Even in this case, CBC can still generate a lot of the same chunks with the original file. In this figure, $p'_{1,1} = f_{i,0}$ and $p'_{1,2} = f_{i,1}$. We show how w affects the search quality in section IV.

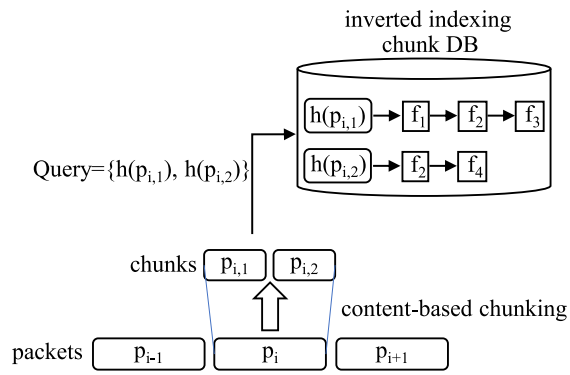

Fig. 4. CBC for packets and naive search.

C. Scalability Problem and Bloom Filter

To overcome the scalability problem of the naive search of Fig. 4, we use a Bloom filter to select only those packets that deserve a search operation. The simple idea is that we collect a set of all chunks while the CBC algorithm generates chunks from the critical files. We use the Bloom filter to remember all the chunks with reduced memory space.

Bloom filters are widely used for not only networking applications but also any computing area [21]. For a given set, a Bloom filter encodes all the members of the set into a bitmap that is initialized to '0'. There are multiple hash functions to find multiple indexes for each element of the set. During the encoding phase, all the elements of the set are recorded in the bitmap by turning the bits indexed by the multiple hash functions to '1'. After the encoding phase finishes, the Bloom filter can be used for membership checking. For any given item, we apply hash functions to have its multiple indexes. If at least one index is of '0', we consider the item is not an element of the set; otherwise, we consider the element may be an element of the set with a certain probability of a false positive that can be computed from the following equation:

$$f_r = (1 - (1 - \frac{1}{m})^{kn})^k \tag{1}$$

where m is the bitmap size, n is the set size, and k is the number of hash functions. In this paper, every chunk from the critical files is an element of the set.

A Bloom filter is built from the chunks of all critical files. This filter selects those packets whose chunks become elements of a query set to the inverted index. Most packets normally do not include chunks of critical files because traffic related with malware download or data leakage is negligibly small, compared with normal traffic volume. Therefore, the Bloom filter prevents unnecessary queries by orders of magnitude.

D. False-Positive Problem and Delayed Query

We present a delayed-query strategy to mitigate the false-positive problem and to reduce queries to the inverted index, which is called Packet-level Bloom-filter & similarity search (PBS) in this paper. The idea is that for a given packet, we keep the numbers of member chunks and all

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 4

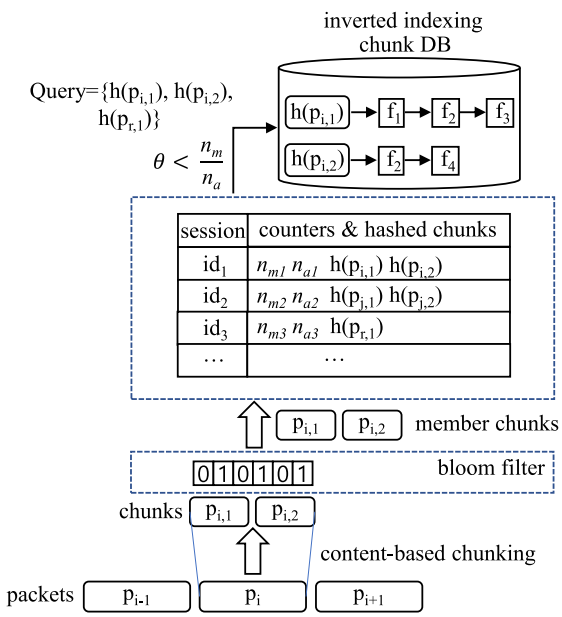

Fig. 5. Packet-level Bloom-filter & similarity search (PBS).

chunks only if at least one chunk from the packet is a member of the Bloom filter. These numbers are accumulated per session. Each session is defined by TCP/IP five tuples of source IP address, destination IP address, source port number, destination port number, and protocol type, denoted as session id. We implement this recording scheme by a simple hash table where the key is the TCP/IP five tuple, or a session id, and the values are two counters, the number of member chunks and the number of all chunks, denoted as n_m and n_a, respectively. The table can be kept concise because only packets that have at least one member chunk are considered for book keeping. Each entry in the hash table consists of a key, two counters, and a set of hashed values of member chunks that belong to the session. Fig. 5 shows the overall architecture of PBS.

Queries to the inverted index is triggered in a delayed way to reduce false-positives and to save computing resources for inverted index search. Query sets are triggered periodically, and the default period is set to five minutes. Only those keys in the table generate query sets only if the ratio of the member-chunk counter to all-chunk counter (n_m/n_a) is above threshold θ. This threshold also reduces false positives. Fig. 5 shows the proposed scheme that has a Bloom filter to check if a given packet has at least one member chunk. Hashed values of member chunks are accumulated in the table together with two counters and the key. The table does not need to be large because the size of hashed values and counters is small.

E. Long Session Problem and Advanced PBS

A file is generally transferred over a session. When the duration time of the session and the file transfer time are almost the same, PBS identifies the file with high accuracy. Fig. 6 (a) shows this case when three packets make a session

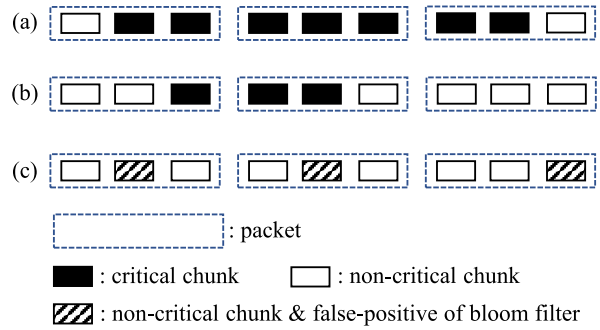

Fig. 6. Limitations of PBS. (a) Normal detection case when $\theta = 0.5$. (b) False-negative case when $\theta = 0.5$. (c) False-positive case when $\theta = 0.3$. Each session simply includes three packets.

and the seven critical chunks are included in the three packets. A critical chunk is defined to be a chunk that belongs to a critical file. On the contrary, the duration time of a session can be much longer than that of a file transfer, which is shown in Fig. 6 (b). For example, a short critical file is transferred within one minute while its session is maintained for longer than five minutes. Other non-critical files can also be transferred. In this case, first two packets pass the bloom filter of PBS, and the table will keep $n_a = 6$ and $n_m = 3$ for this session. If θ were set to be greater than 0.5, PBS would not identify the transfer of this critical file. We call this a long session problem in this paper, which may cause PBS not to identify critical files. This can be called as a false negative because a critical file is not detected mistakenly.

We observe that a false positive, i.e. a non-critical file is detected mistakenly, can occur with PBS. Fig. 6 (c) shows that all the three packets of a session have one false-positive of the bloom filter, respectively. Because each of three packets has one chunk that passes the membership-checking of the bloom filter, PBS keeps $n_a = 9$ and $n_m = 3$ for this session. If θ were set to be smaller than 1/3, PBS would trigger a query to the inverted index to find any critical file. These critical chunks from the packets are incorrectly identified by accident, and therefore no critical file in the inverted index would match the query. However, the query itself brings a big burden on the inverted index system because the number of invalid queries is proportion to the number of packets that is usually an enormous number. This problem may be mitigated if the false-positive rate of a bloom filter is minimized. However, a bloom filter can save more space when a larger false-positive rate is tolerated. Therefore, the false-positive rate cannot be lowered indefinitely.

In this paper, we present an advanced PBS, APBS, to mitigate both false-negative and false-positive problems described above. Two threshold values are adopted in APBS at the different levels of a bloom filter and a query to inverted index. The idea is to count the number of critical chunks that belong to a critical file with a high probability, and a query to inverted index is launched only when a sufficient number of those critical chunks is gathered.

The first threshold, θ_c, is applied to the bloom filter, which allows only those packets that have more than θ_c critical chunks to pass the bloom filter. On the contrary, a packet

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 5

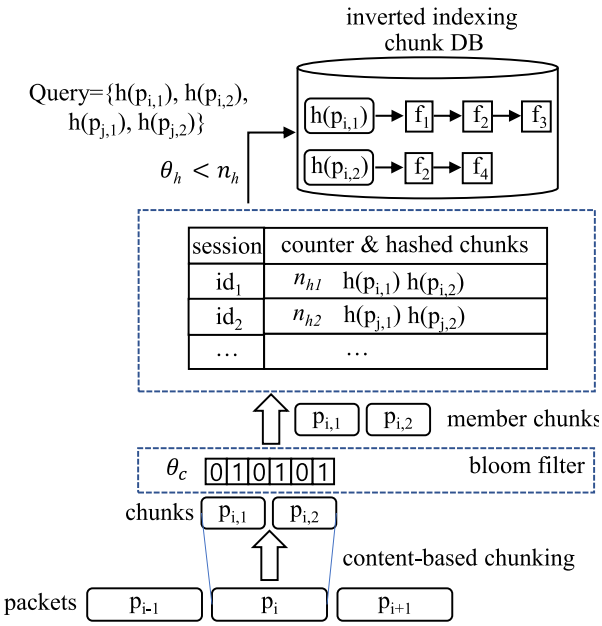

Fig. 7. Advanced PBS with two threshold values of θ_c and θ_h that mitigate the false positives and false negatives of PBS, respectively.

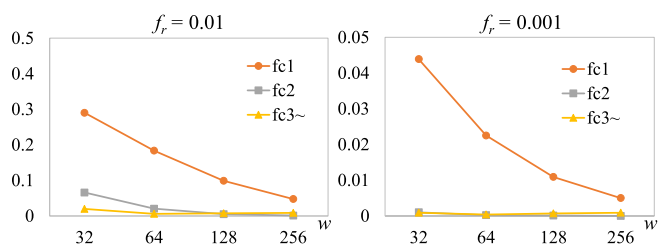

Fig. 8. Comparison of probabilities that one, two, or more than two false-positive chunks occur at the same time in a packet of 1,500 bytes, denoted as $fc1$, $fc2$, and $fc3\sim$, respectively, as the window size changes from 32 to 256.

can pass the bloom filter when it has any critical chunk in PBS. In APBS, the number of critical chunks per session is accumulated that are included in the packets that have passed the bloom filter. In this sense, the number of critical chunks per session except the false positives of the bloom filter is maintained in the hash table as in Fig. 7. This first threshold reduces false positives at the level of a bloom filter. The non-critical but false-positive chunks in Fig. 6 (c) can be ignored by θ_c. We emphasize that the number of false positives can be large even if the false-positive rate of the bloom filter is small. This is because the number of packets per session can be very large.

The second threshold, θ_h, controls the query to search the inverted index. When a packet passes the bloom filter, the counter of the critical chunks per session, n_h, increases by the number of critical chunks inside the packet. When n_h is greater than θ_h, a query is launched to search the inverted index as in Fig. 7. This counter of APBS significantly improves the periodic and delayed query of PBS, which makes a prompt query. The most important effect is that θ_h can reduce false negatives; a false negative may occur in PBS as shown in Fig. 6 (b) when n_m/n_a is less than or equal to θ and no query is triggered. On the contrary, a query is always triggered when n_h becomes greater than θ_h. This solves the long session problem of PBS.

Last but not least, θ_c and θ_h also improve the processing speed and refine queries in APBS. A query to the inverted index takes significant time and computing resources in finding the most relevant critical file to the query. In PBS, packets pass the bloom filter by accident with the probability of f_r of equation (1). In APBS, a packet can be considered for the query only if it has more than θ_c critical chunks at the same time. Besides, the number of accumulated critical chunks should be greater than θ_h to generate a query. Fig. 8 shows how θ_c effectively reduces those packets that have only one false-positive chunk. The figure shows the probability that one false-positive chunk can occur in a packet, denoted as $fc1$, two false-positive chunks as $fc2$, more than two false-positive chunks as $fc3\sim$. The x-axis is the average window size. We assume that a packet size is 1,500 bytes. We set f_r to 0.01 and 0.001 for the left and right plots, respectively. We emphasize that one packet hardly includes two or more false-positive chunks at the same time by accident while one false-positive chunk may occur. On the contrary, we observe that critical packets are filled with many critical chunks, which motivates us to develop APBS.

TABLE I
NUMBER OF INDEXED AND TEST (TRANSFERRED) FILES

type	wiki	malware
inverted index	980	1,000
test (identical)	100	100
test (similar)	100	100
test (different)	100	100

IV. EXPERIMENTS

We evaluate our proposed scheme through extensive experiments using real datasets of document files and malware files. We implement PBS and APBS on the Elasticsearch platform. The experimental results show that PBS can successfully identify a critical file from packets except long sessions. On the contrary, APBS can identify any critical file even in long sessions while false positives and false negatives can be reduced to zero.

A. Experimental Setup

Two datasets are built for the experiments. For each dataset, we first build an inverted index and a Bloom filter from critical files after the CBC algorithm is applied to each file. Then, test files are transferred over the network and packets are captured to identify the most relevant critical file from the inverted index. The set of test files consist of three different types, identical to the indexed files, similar to the indexed files, and different to the indexed files. The similarity between two files is measured by the jaccard index between their chunk sets. Table I summarizes the number of files used for the experiments.

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 6

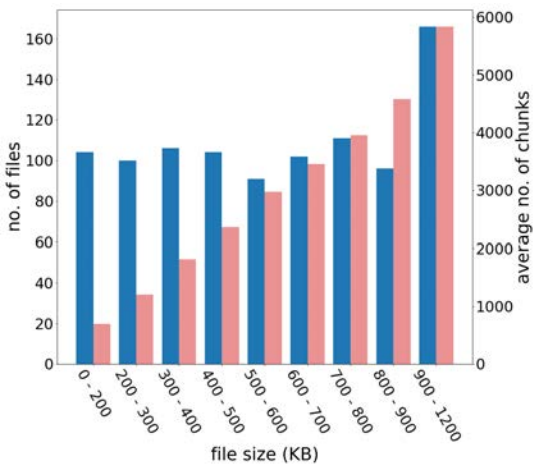

Fig. 9. Distribution of indexed wiki documents and their average chunk size.

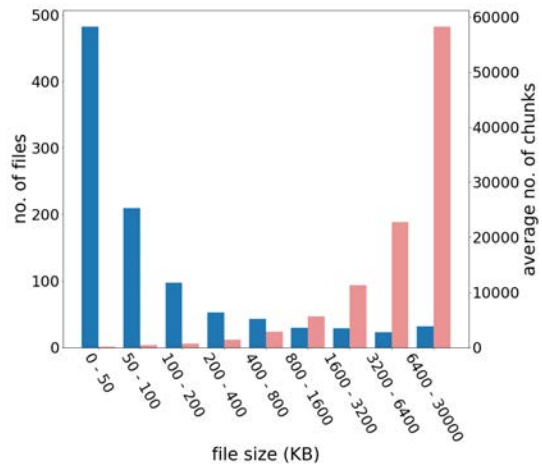

Fig. 10. Distribution of indexed malware files and their average chunk size.

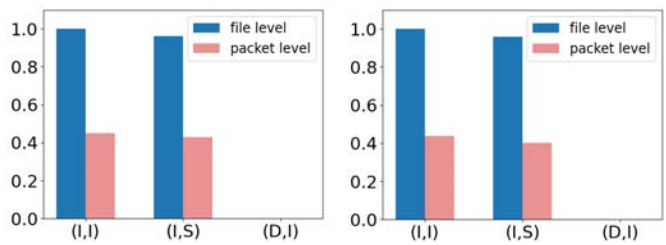

Fig. 11. Average jaccrad index among *Identical*, *Similar*, and *Different* groups. Left is Wiki and right is Malware.

The first dataset includes 980 documents downloaded from Wikipedia [22]. We extract contents-based chunks from each document and build an inverted index. We select 100 documents from 980 indexed files, and download each of the old versions of them from Wikipedia. Therefore, we obtain 100 documents whose similar files are stored in the inverted index. We call these 100 documents as similar wiki files. We also obtain another 100 documents that are not related with the 980 indexed documents or 100 similar files, which is called different wiki files. Finally, we select another 100 documents from the indexed documents, which are called identical wiki files. We transfer 100 identical wiki files, 100 similar wiki files, and 100 different wiki files over the network during the experiments. Fig. 9 shows the distribution of indexed wiki files and their average number of chunks.

The second dataset includes 1,000 PE malware files downloaded from the VirusShare [23]. We extract contents-based chunks from each malware file and build an inverted index. We select 100 malware files from 1,000, and search our malware database for another malware files that are similar to the selected 100 malware files in terms of the jaccard index between their chunk sets. In this way, we obtain 100 malware files whose similar files are stored in the inverted index. We call these 100 malware files as similar malware files. We also obtain another 100 benign PE files that are not related with the 1,000 indexed malware files or 100 similar malware files, which are called different benign files. Finally, we select another 100 malware files from the indexed malware files, which are called identical malware files. We transfer 100 identical malware files, 100 similar malware files, and 100 different benign files over the network during the experiments. Fig. 10 shows the distribution of indexed malware files and their average number of chunks.

We compute the average jaccard index between two test file groups summarized in Table I. We measure the jaccard index at file level and packet level with the AE chunking algorithm of $w = 128$. Fig. 11 shows the average jaccard index. The file-level jaccard index of (I,I) is 1, but the packet-level is less than 0.5. This is because at least the first and last chunks of a packet do not match the file chunks as shown in Fig. 2. The file-level jaccard index of (I,S) is high because we select very similar files to the critical files for the experiments.

Three different network protocols are used for experiments of file transfer. The first protocol is the standard file transfer protocol (ftp) [24]. The second protocol is the file transfer API provided by a commercial network attached storage (NAS) [25]. We capture NAS packets and confirm that the base protocol is hypertext transfer protocol (HTTP), but the protocol seems customized. The third protocol is developed by us with simple C socket programming (socket) to transfer files. We observe that three protocols use TCP over Ethernet. Files are transferred between two machines and there are three network switches between them.

B. File Transfer and Packet Capture

When wiki documents and malware files are transferred over the network, we capture their packets between client and server machines by using Linux tcpdump 4.9.3 on libpcap 1.8.1. Note that 300 test documents and 300 test PE files are transferred as described above. Capturing packets without loss is a challenging problem and many researchers have presented solutions. In this paper, we use a packet-capture library, but sophisticated capturing technologies are beyond the scope of this paper. We stress that our proposed scheme can identify which critical file is related with a given packet, or a group of packets. This can identify which session is related with

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 7

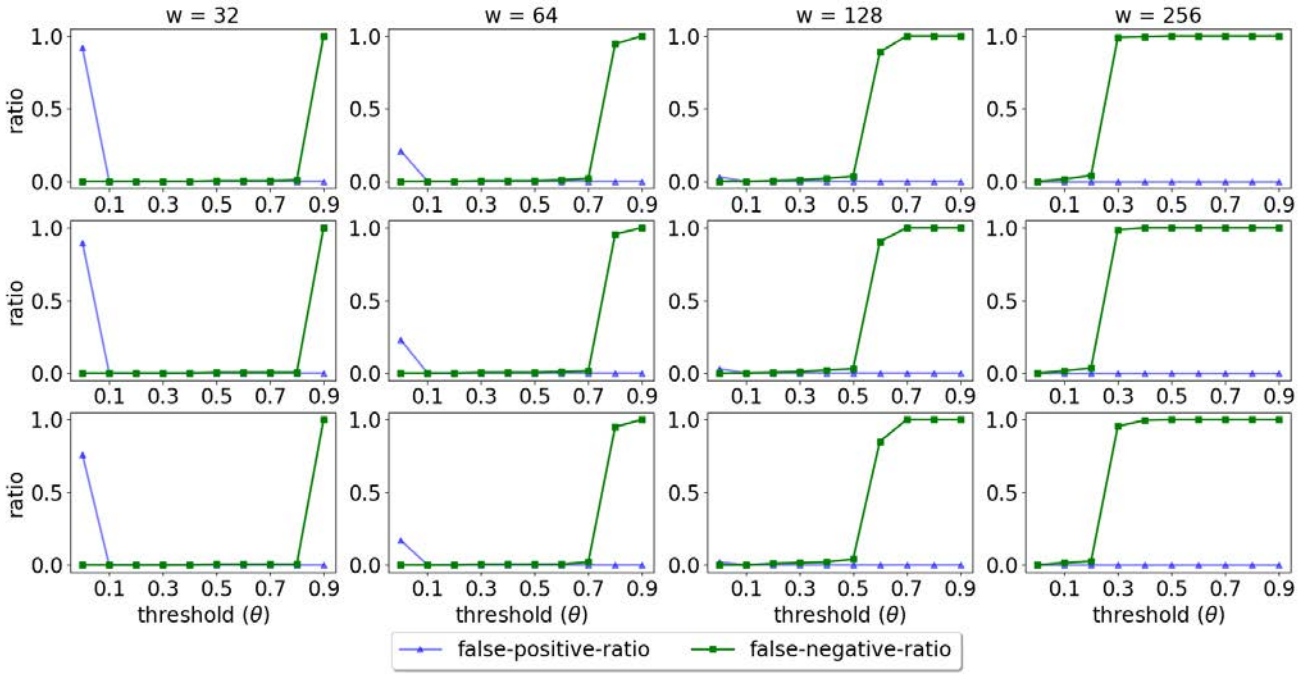

Fig. 12. Comparison of PBS false-positive and false-negative ratios with the wiki dataset. The first, second, and third rows are the results from ftp, NAS, and socket programming, respectively. The target false-positive rate of the Bloom filter is 0.001.

transferring critical files even when a few packets in the session are missed during packet capturing.

For comparison and test purposes, we install Zeek [6], an open-source network security monitoring tool, and run it as a default configuration. Zeek reassembles packets of the same session and extract files embedded in the session. We observe that Zeek can successfully extract files only for ftp. Files are not extracted from other two protocols by Zeek, which is a well-known limitation. On the contrary, our proposed scheme successfully identifies which indexed critical file matches packets. We also observe that Zeek fails to restore files from packets if more than one packet from the session is missed during packet capturing. Our proposed scheme solves this problem as well because a critical file can be identified by its partial byte streams.

C. Experimental Results of PBS

We first evaluate experimental results of PBS by false-positive and false-negative rates. In this paper, a captured packet causes one of four results, true positive, true negative, false positive, and false negative. Suppose that a packet is captured while an identical or similar file is transferred over the network. A true positive occurs when the critical file is exactly identified from this packet; otherwise, a false negative occurs. On the contrary, suppose that a packet is captured while an different file is transferred over the network. A true negative occurs when no critical file is identified from this packet; otherwise, a false negative occurs. The false-positive rate is the ratio of the number of false positives to the total number of false positives and true negatives. The false-negative rate is the ratio of the number of false negatives to the total number of false negatives and true positives.

In the first set of experiments, we test PBS with wiki documents. For each of 980 critical files, we apply the AE chunking algorithm with four different average chunk size, $w = 32$, 64, 128, and 256. Note that different w generates different chunks. When measuring the jaccard index of two files, we use their chunk sets of $w = 128$. The numbers of total distinct chunks are 7,868,665, 4,897,387, 2,856,389, and 1,548,613, for $w = 32$, 64, 128, and 256, respectively. Reflecting chunk numbers, we implement the Bloom filter of false-positive rate 0.001. The bitmap size and the number of hash functions of each Bloom filter becomes different with w.

Next, we transfer 100 identical, 100 similar, and 100 different test files over the network. Three network protocols are used one by one to verify PBS strictly. During the file transfer, packets are captured and processed by PBS to identify critical files. Fig. 12 plots experimental results from four different w values and three protocols. The first row shows the experimental results of ftp; from left to right, we show the experimental results with $w = 32$, 64, 128, and 256. Each plot compares the false-positive and false-negative ratios when θ changes from 0 to 0.9.

We discuss the experimental results from Fig. 12. First, three protocols show similar patterns. We stress that Zeek, the current best practice, can identify critical files from packets only when ftp is used. This is because only well-known protocols can be interpreted. On the contrary, PBS successfully identifies critical files over all protocols. Second, the false-negative ratio becomes worse with larger w. A large w splits a file and especially a packet into a small number of large chunks. For example, a packet payload of 1,400 bytes is considered large. However, only 5 chunks are generated on average when $w = 256$. Because first and last chunks from

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 8

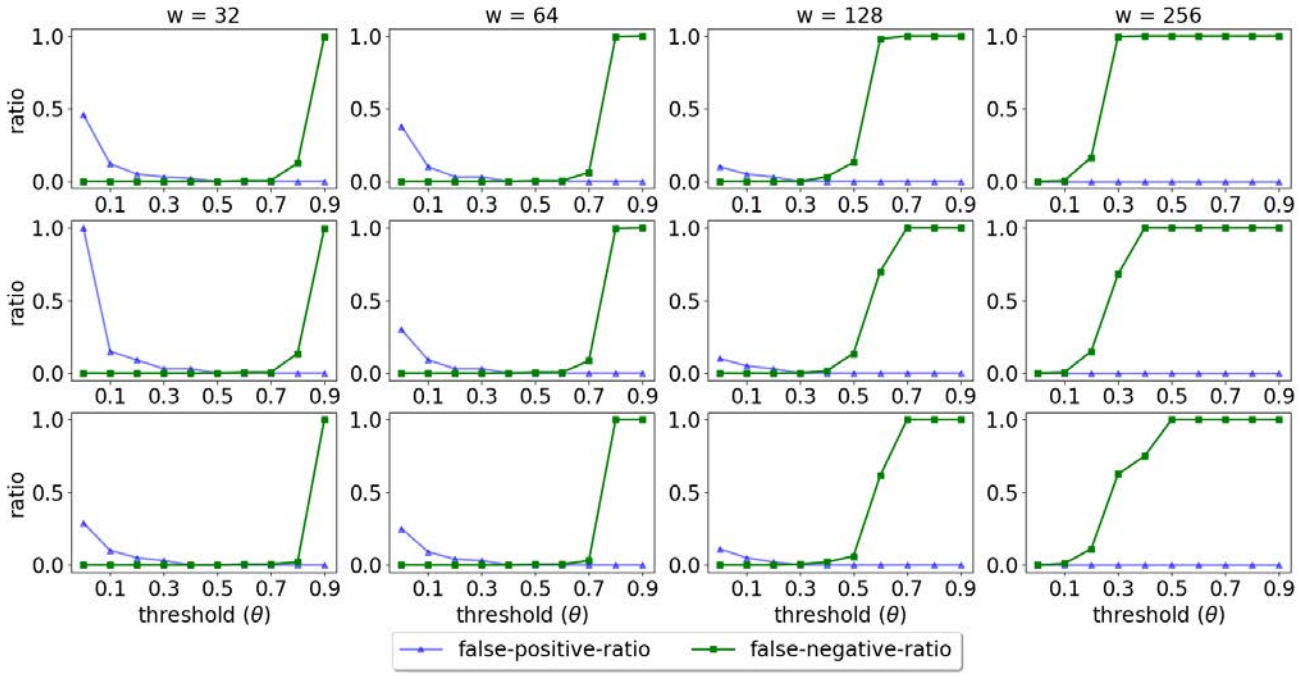

Fig. 13. Comparison of PBS false-positive and false-negative ratios with the malware dataset. The first, second, and third rows are the results from ftp, NAS, and socket programming, respectively. The target false-positive rate of the Bloom filter is 0.001.

a packet would be different from the chunks from even the same file, the false-negative ratio increases. The plots show that large w would reduce the false-positive ratio, but the false-negative ratio increases. When w becomes small, the false-positive ratio increases. Besides, small w generates more distinct chunks, which requires more system resources for building the inverted index and search operations. We observe that $w = 32$ generates 7,868,665 distinct chunks from the wiki dataset while $w = 256$ generates only 1,548,613 chunks. Optimal θ values for PBS can be found from Fig. 12 that make both false-positive and false-negative ratios zero. This means that no packets cause any error in identifying critical files. Another advantage of PBS is that the number of queries to the inverted index is minimized. In our experiments, one query is enough for one session in most cases. We believe that this high accuracy and light operations make PBS feasible for real-time detection of critical files from packets.

In the second set of experiments, we test PBS with malware files. Although byte streams of the malware files have higher randomness than wiki documents, the overall patterns of experimental results are similar to those of wiki documents. We observe that $w = 32$ generates 11,739,785 distinct chunks from the malware dataset while $w = 256$ generates only 1,739,548 chunks. Fig. 13 plots the experimental results for different w values and three protocols. Multiple optimal θ values can be found here that make both false-positive and false-negative ratios zero. From Figs. 12 and 13, PBS performs best at $w = 64$ and $\theta = 0.4$ where the numbers of distinct chunks are 4,897,387 and 6,130,003 for wiki and malware datasets, respectively.

There are two main parts for PBS that determines the performance, a Bloom filter and an inverted index. The index can be built on high-end servers and the Elasticsearch platform proves its scalable performance in a range of industries. However, the Bloom filter should be kept small and a smaller number of hash functions is preferable because every packet is checked against the Bloom filter. We repeat our experiments with a light Bloom filter setting. We change its target false-positive rate from 0.001 to 0.1, which means that the number of hash functions are reduced from 10 to 3, and the memory space is also reduced in the same proportion. Fig. 14 shows the experimental results on wiki documents when the light Bloom filter is used. The plots show that $w = 256$ would be a better choice for PBS when a light Bloom filter is used. In this case, the number of distinct chunks to build the inverted index is reduced to 1,548,613. The disadvantage of the small Bloom filter is that more chunks would become false positives and they occupy the memory space of the hash table. Fig. 15 shows the experimental results on malware files when the light Bloom filter is used.

The limitations of PBS are shown in Fig. 16 where the long session problem causes PBS not to identify critical files. In this experiment, we transfer one critical file together with multiple non-critical files. In NAS and socket programming, the second and third rows in the figure, false-negatives occur because a query to the inverted index is not triggered due to the low value of n_m/n_a. False positives of the bloom filter increases n_a. On the contrary, the first row of the figure, ftp, still successfully identifies the critical file because one session is dedicated to one file transfer in ftp. However, PBS fails in the other protocols, which motivates us to invent APBS.

D. Experimental Results of APBS

We perform a set of experiments that show APBS can identify critical files over a long session with multiple non-critical files. We emphasize that PBS fails in the same setting

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036

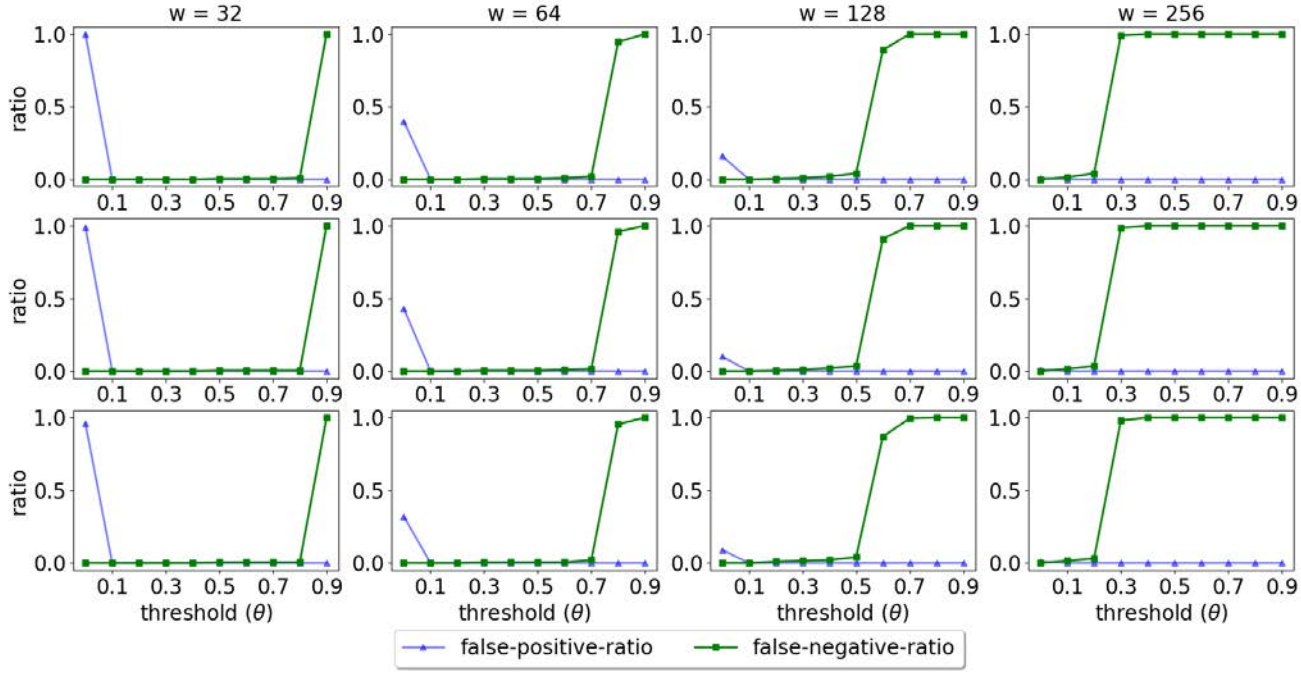

Fig. 14. Comparison of PBS false-positive and false-negative ratios with the wiki dataset. The first, second, and third rows are the results from ftp, NAS, and socket programming, respectively. The target false-positive rate of the Bloom filter is 0.1.

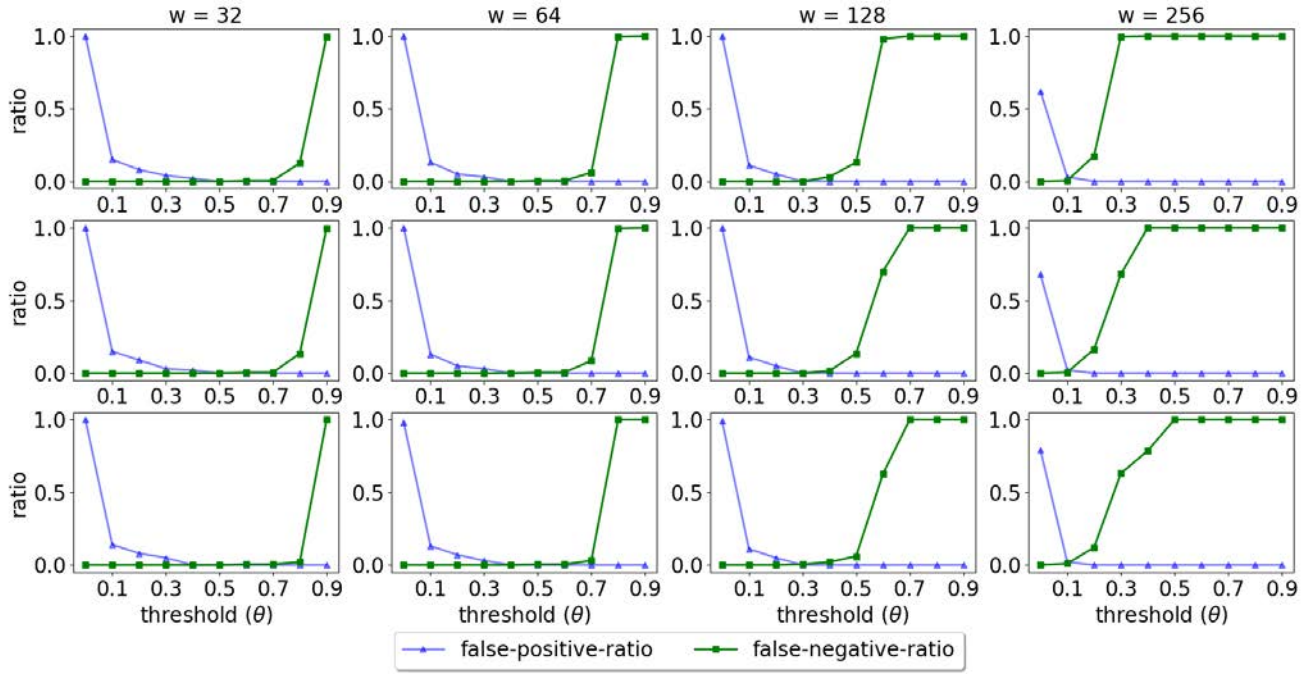

Fig. 15. Comparison of PBS false-positive and false-negative ratios with the malware dataset. The first, second, and third rows are the results from ftp, NAS, and socket programming, respectively. The target false-positive rate of the Bloom filter is 0.1.

because false negatives occur. Fig. 17 plots the false-positive and false-negative ratios of APBS for the three protocols of ftp, NAS, and socket programming. For each protocol, we run APBS to identify critical files while changing θ_h to 1, 4, 16, and 64. The x-axis is θ_c that controls the number of packets passing the bloom filter. When θ_h is set to 1, false-positives occur in all the three protocols; a large number of queries are launched. When θ_h is 64, the queries are restricted, and therefore the false positives decrease.

In Fig. 17, false positives decrease and false negatives increase as θ_c increases. This is because fewer packets can pass the bloom filter by accident. It is interesting that three protocols show a similar pattern. We argue that APBS can stably identify critical packets irrespective of a protocol type.

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 10

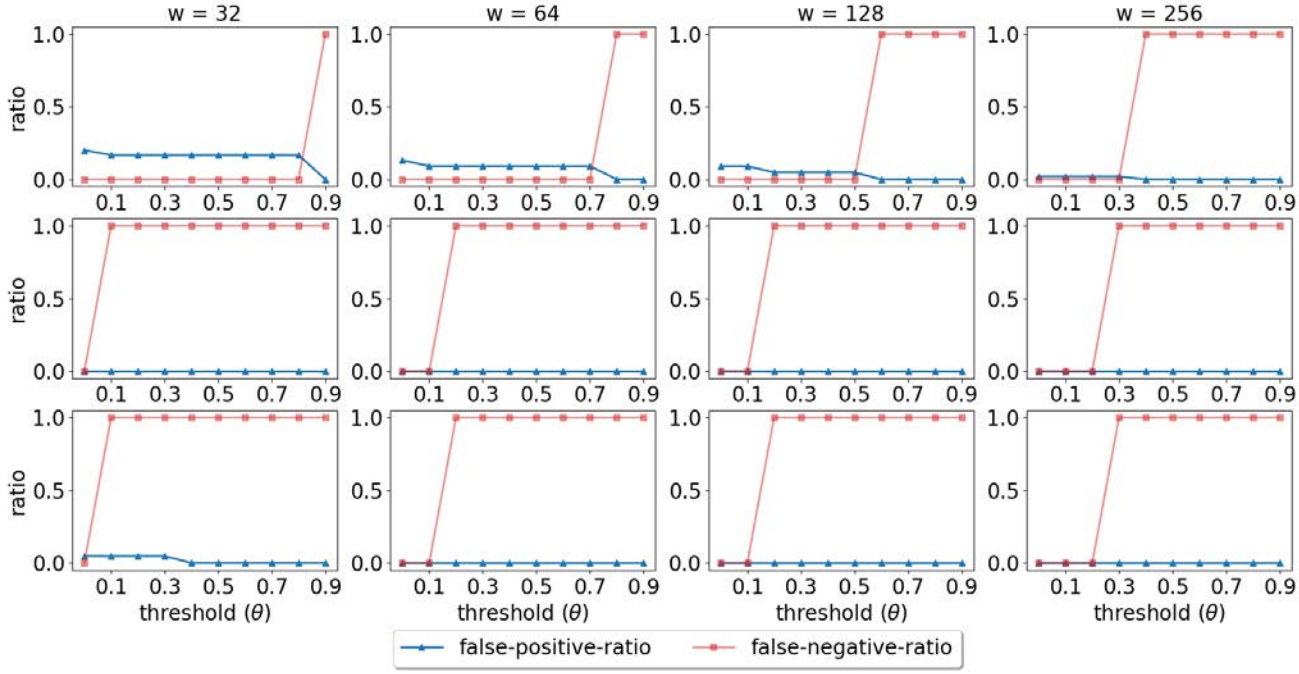

Fig. 16. Long session problem of PBS. The false-negative ratios of NAS and socket programming, the second and third rows, become worse when the session includes one critical file and multiple non-critical files.

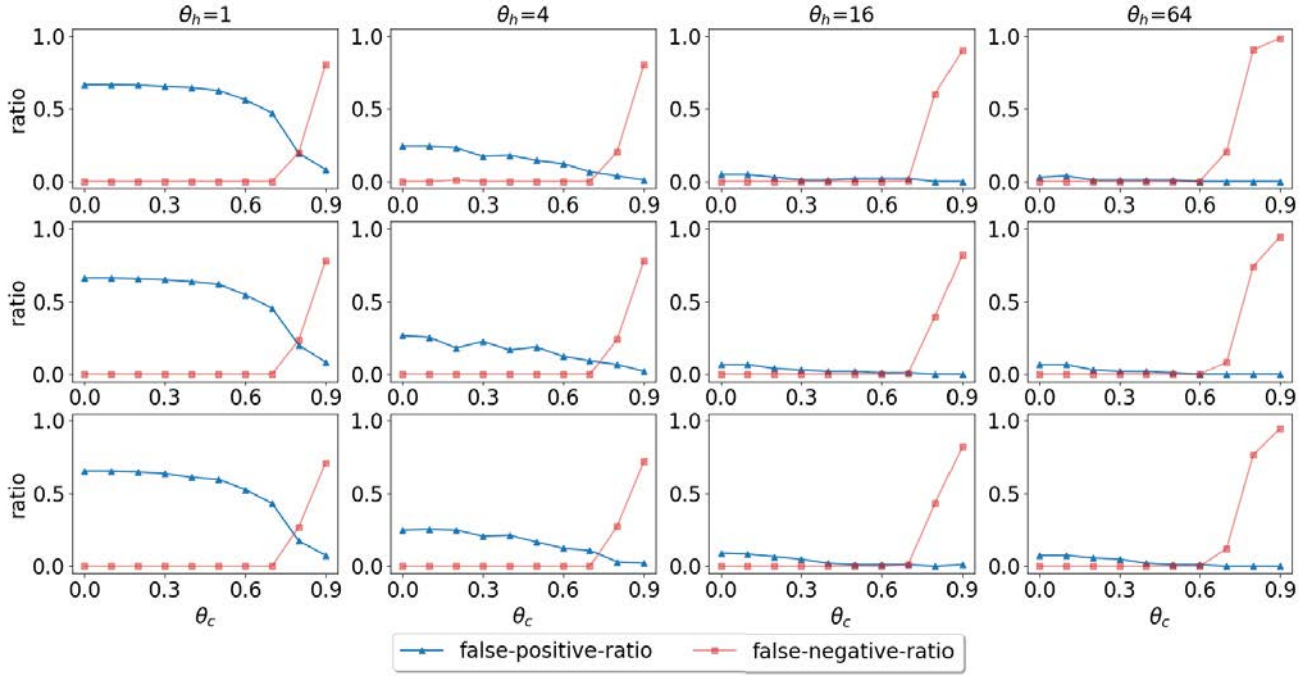

Fig. 17. Comparison of APBS false-positive and false-negative ratios with wiki dataset. The plots are results from ftp, NAS, and socket programming, respectively. The target false-positive rate of Bloom filter is 0.1 and the average window size is 128.

We repeat the same experiments on the malware dataset, and the result is shown in Fig. 18. There is no big difference between two datasets.

There is a trade-off between the accuracy and processing time for our proposed scheme, which is certainly affected by w. Only accuracy-related metrics have been presented in Figs. 12∼18. We compare the processing times of APBS with different w values in Fig. 19 where 300 test files of wiki documents are transferred over ftp, the same as the previous experiment, and 96,943 packets are captured. We measure the total processing time that includes packet chunking, membership checking of the bloom filter, hash table processing, queries to Elasticsearch search and replies. We confirm that the processing times decrease almost linearly with w as shown

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 11

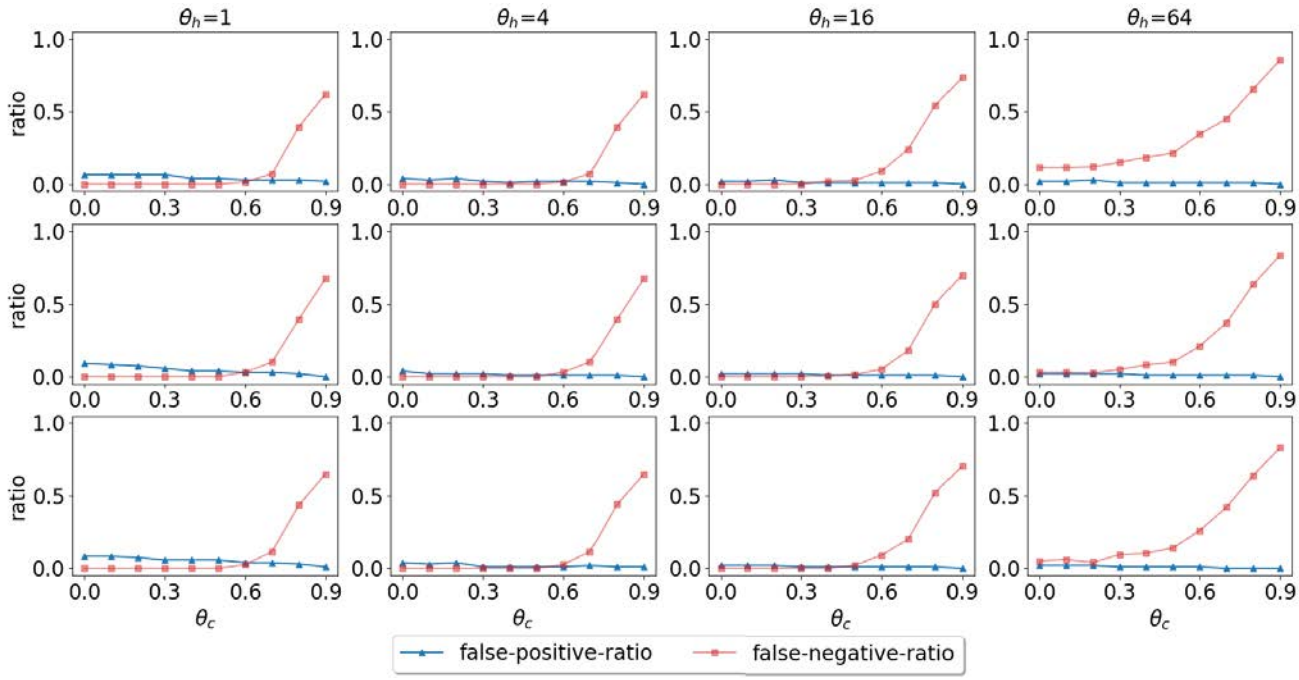

Fig. 18. Comparison of APBS false-positive and false-negative ratios with malware dataset. The plots are results from ftp, NAS, and socket programming, respectively. The target false-positive rate of Bloom filter is 0.1 and the average window size is 128.

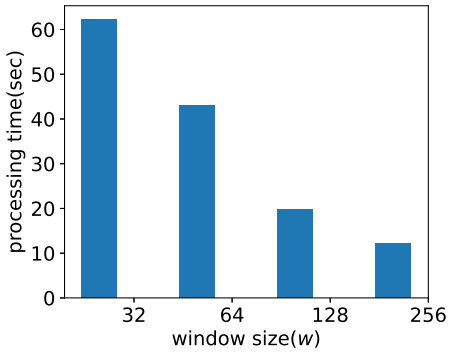

Fig. 19. Comparison of processing time of APBS with different w's. The wiki dataset is transferred over the ftp with θ_c and θ_h set to 0.5 and 16, respectively.

in Fig. 19. This is not surprising because w is directly related with the number of chunks.

E. Discussion

We discuss the experimental results, contributions, and limitations of PBS and APBS. Considering Figs. 12∼19 together, there is a trade-off between the accuracy and the processing time in terms of w. Smaller w may improve the false-positive and false-negative, but degrade the processing time; smaller w also generates more chunks that should be recorded in bloom filters and indexed in the Elasticsearch, which may make the system heavy. Through our exhaustive experiments, we find a rule of thumb that both the accuracy and the processing time would be balanced with $w = 128$.

After a critical file is indexed, the proposed scheme can find its similar files transferred over multiple packets. Therefore, if a malware file is indexed a priori, its similar files can be identified. Actually, recent studies reveal that redundant malware files are frequently observed [4], [15], [16]. On the contrary, a new malware file cannot be identified by PBS and APBS because the file has not been obtained and no indexing was established. We emphasize that PBS and APBS can find similar files over packets instead of detecting totally new malware types.

Both PBS and APBS inspect packets and make chunks from them. Although encrypted packets cannot be efficiently inspected, still a significant number of malware infection or inadvertent data infiltration incidents are delivered in non-encrypted packets [2], [17]. If necessary, an additional decryption box can be deployed to obtain plain packets from encrypted ones [26]. Recently, anomaly detection for encrypted packets is also studied [27], [28]. The main limitation of PBS and APBS would be that encrypted packets cannot be inspected and therefore critical files cannot be identified if no decryption boxes are supported. Actually, this limitation is common in any network-based security solutions, which is beyond the topic of this paper [6], [7].

V. Related Work

According to IBM's 2016 Cost of Data Breach Study, the average consolidated cost of a data breach has reached $4 million. A data breach can be the intentional or inadvertent exposure of confidential information to unauthorized parties [3], [29]. Inadvertent insiders send confidential documents or their previous versions via emails without encryption, and they often save the digital copies at mobile devices. It is not surprising that they transfer the files as plain texts by mistakes. Encrypted packets cannot be directly inspected by

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 12

network-based security devices. Researchers study decryption boxes [26] or encrypted packet analysis [27], [28].

Recent studies show that malware variants are prevalent because developing new malware from the scratch is a challenging task to attackers and malware-as-a-service provides variant malware samples easily to users. Ugarte-Pedrero et al. reveal that few malware samples are unique and the number of similar malware files increases steadily [4]. The similarity of malware files can be defined differently; they may run similarly by calling the same API call sequence, they may try to contact the same C2 servers, analysis reports by automatic tools are similar, or their byte sequences as a file are similar. We are interested in the last case in this paper. It is interesting that some benign files also show strong similarity except a few bytes because of different license information or code optimization [14].

There are open source network monitoring devices that can extract files from packets [6], [7]. They first reassemble packets of the same session, which consumes huge computing resources. Only when the protocol can be interpreted by these devices, an embedded file can be extracted. Breitinger and Baggili proposed a Bloom-filter based scheme to determine if a packet's payload has common contents with any of the critical files [17]. Their work is closely related with this paper. However, their scheme is for membership check only and does not identify a specific critical file.

A number of fuzzy hashes are proposed to find similar files [5], [17], [30]. The main idea is to divide a file into multiple parts and each part is encoded into a small string or bitmap. If two files are similar, some parts of the string or bitmap would be identical. However, fuzzy hashes require inefficient pairwise comparisons. Inverted indexing can relax this problem if each key appears in a limited number of files [10]. Elasticsearch is an open source platform to provide inverted indexing in distributed systems [11].

Contents-based chunking is related with fuzzy hashing. Manber uses a rolling hash to find dynamic cutting points based on partial file contents [31]. These partial identical signatures make it possible to compare two similar files. Because network packets also have redundant payloads, replacing frequent byte streams with a small hash value can save network bandwidth. This motivates many researchers to study packet-level contents-based chunking algorithms [8], [9], [19], [32].

VI. Conclusion

In this paper, we present a new critical file identification scheme by inspecting a packet or a group of packets. This new scheme does not require packet reassembly, which saves huge amounts of system resources in deep packet inspection. A content-based chunking algorithm consistently splits files and packets into chunks, and hashed values of the chunks become keys for the inverted indexing. The proposed scheme uses a light Bloom filter to select only those chunks that deserve an expensive search operation, which saves system resources by orders of magnitude. The new scheme can identify critical files even when they or their similar files are transferred over unknown protocols. We also present an advanced scheme that not only identifies critical files even in a challenging environment but also reduce unnecessary queries by orders of magnitude. Intrusion detection systems and data leakage prevention systems can adopt this new scheme, for example in detecting malware variants download and confidential data infiltration by inadvertent insiders.

References

[1] J. Hur, H. Jeon, H. G. Shon, Y. J. Kim, and M. Yoon, "Finding critical files from a packet," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, May 2021, pp. 1–10.

[2] Sophos News. (2020). *Nearly a Quarter of Malware Now Communicates Using TLS*. Accessed: Feb. 28, 2022. [Online]. Available: https://news.sophos.com/en-us/2020/02/18/nearly-a-quarter-of-malware-now-communicates-using-tls/

[3] Y. Shvartzshnaider et al., "Vaccine: Using contextual integrity for data leakage detection," in *Proc. World Wide Web Conf.*, 2019, pp. 1702–1712.

[4] X. Ugarte-Pedrero, M. Graziano, and D. Balzarotti, "A close look at a daily dataset of malware samples," *ACM Trans. Privacy Secur.*, vol. 22, no. 1, pp. 1–30, Jan. 2019, doi: 10.1145/3291061.

[5] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digit. Invest.*, vol. 3, pp. 91–97, Sep. 2006.

[6] Zeek. (2022). *An Open Source Network Security Monitoring Tool*. Accessed: Feb. 28, 2022. [Online]. Available: https://zeek.org

[7] Suricata. (2022). *Open source IDS/IPS/NSM Engine*. Accessed: Feb. 28, 2022. [Online]. Available: https://suricata-ids.org/

[8] Y. Zhang et al., "AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1337–1345.

[9] S.-H. Shin et al., "PrIDE: A protocol-independent de-duplication engine for packet recording," *IEEE Netw.*, vol. 30, no. 6, pp. 42–48, Nov./Dec. 2016.

[10] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[11] Elasticsearch. (2022). *Get Started With Elasticsearch*. Accessed: Feb. 28, 2022. [Online]. Available: https://www.elastic.co/

[12] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[13] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," in *Internet Mathematics*. Princeton, NJ, USA: Citeseer, 2002.

[14] B. Li, K. Roundy, C. Gates, and Y. Vorobeychik, "Large-scale identification of malicious singleton files," in *Proc. ACM CODASPY*, Mar. 2017, pp. 227–238.

[15] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. 28th USENIX Secur. Symp. (USENIX Security)*. Santa Clara, CA, USA: USENIX Association, Aug. 2019, pp. 729–746.

[16] H. Aghakhani et al., "When malware is packin' heat; limits of machine learning classifiers based on static analysis features," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, 2020, pp. 1–20.

[17] F. Breitinger and I. Baggili, "File detection on network traffic using approximate matching," *J. Digit. Forensics, Secur. Law*, vol. 9, p. 3, Sep. 2014.

[18] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compress. Complex. SEQUENCES*, 1997, pp. 21–29.

[19] Y. Zhang and N. Ansari, "On protocol-independent data redundancy elimination," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 455–472, 1st Quart., 2014.

[20] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, nos. 23–24, pp. 2435–2463, Dec.1999.

[21] M. K. Yoon, J. Son, and S.-H. Shin, "Bloom tree: A search tree based on Bloom filters for multiple-set membership testing," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1429–1437.

[22] Wikipedia. (2022). *Welcome to Wikipedia*. Accessed: Feb. 28, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Main_Page

[23] (2022). *VirusShare*. Accessed: Feb. 28, 2022. [Online]. Available: https://virusshare.com/

[24] *File Transfer Protocol*, document RFC 959, Oct. 1985. [Online]. Available: https://www.rfc-editor.org/info/rfc959

[25] (2022). *Synology*. Accessed: Feb. 28, 2022. [Online]. Available: https://www.synology.com/

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.

[26] J. Fan, C. Guan, K. Ren, Y. Cui, and C. Qiao, "SPABox: Safeguarding privacy during deep packet inspection at a MiddleBox," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3753–3766, Dec. 2017.
[27] E. Areström and N. Carlsson, "Early online classification of encrypted traffic streams using multi-fractal features," in *Proc. IEEE INFOCOM*, Apr. 2019, pp. 84–89.
[28] T. van Ede et al., "FlowPrint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in *Proc. NDSS*, vol. 27, 2020, pp. 1–18.
[29] L. Cheng, F. Liu, and D. D. Yao, "Enterprise data breach: Causes, challenges, prevention, and future directions," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 7, no. 5, p. e1211, Jun. 2017.
[30] T. Göbel, F. Uhlig, and H. Baier, "Evaluation of network traffic analysis using approximate matching algorithms," in *Proc. IFIP Int. Conf. Digit. Forensics*. Cham, Switzerland: Springer, 2021, pp. 89–108.
[31] U. Manber, "Finding similar files in a large file system," in *Proc. USENIX Winter*, 1994, pp. 1–10.
[32] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proc. ACM SIGCOMM*, 2000, pp. 87–95.

Junnyung Hur received the B.S. degree in computer engineering from Kookmin University, South Korea, in 2019, where he is currently pursuing the master's degree. His current research interests include cyber security, big data platform, and data analysis.

Hyeon Gy Shon received the B.S. and M.S. degrees in computer engineering from Kookmin University, Seoul, South Korea, in 2020 and 2022, respectively. He currently works at MUHAYU, Seoul, as an NLP Research Engineer. His research interests include natural language processing, semantic code searching, paraphrase detection, and artificial intelligence for natural language processing.

Young Jae Kim received the B.S. and M.S. degrees in computer engineering from Kookmin University, Seoul, South Korea, in 2019 and 2021, respectively. He currently works at Common Computer, Seoul, as an AI Research Engineer. His research interests include a large scale deep-neural networks, transformer learning, few shot learning, and prompt engineering.

Myungkeun Yoon (Member, IEEE) received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, South Korea, in 1996 and 1998, respectively, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2008. He worked for the Korea Financial Telecommunications and Clearings Institute, Seoul, from 1998 to 2010. He has been a Professor with the Department of Computer Engineering, Kookmin University, Seoul, since 2010. His research interests include computer and network security, security intelligence, data science for cyber security, and artificial intelligence for security.

Authorized licensed use limited to: Nathan Lowenstein. Downloaded on September 06,2024 at 17:06:57 UTC from IEEE Xplore. Restrictions apply.
Security First Innovations, LLC, Exhibit 2036
Page 2036 - 14